Week 6 Day 3

Spring Data and ORMs

Spring Data

REVATURE

Family of frameworks for ORM and data persistence

- Specifically using Spring Data JPA
- Uses JPA and Hibernate behind the scenes
- Allows us to implement data layer extremely quick
- Abstracts away hibernate and JPA, giving us implementation for more common DAO methods

Object Relation Mapping

REVATURE

Converts data between relational databases and object-oriented programming languages

- Map objects to a table
- Hide details of sql from code
- Automatic versioning and time stamping
- Good for large projects
- Injected transaction management
- Configurable logging
- Faster development

Spring Data JPA: Java Persistence API

REVATURE

Spring Data Module that uses the Java Persistence API

- Decreases time to implement data layers
- Gives implementation for DAO methods from method stubs
- JPA is the standard Java API for managing objects in databases
- Found in javax.persistence
  - Uses JPQ language for querying objects
  - Uses EntityManager for CRUD

Hibernate

REVATURE

ORM Tool for Java Programming

- Used behind the scenes for Spring Data JPA
- Flexible and powerful ORM solution
- Defined under org.hibernate
- Uses HQL to query objects
- Uses Hibernate Sessions to make CRUD

Hibernate Annotation

REVATURE

- @Entity
- @Table
- @Column
- @Id
- @ManyToMany
- @ManyToOne
- @OneToMany

```java
@Entity
@Table(name="users")
public class User {

	@Id
	@Column(name="user_id")
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	@Column(name="first_name", nullable=false)
	private String firstName;

	@Column(name="last_name", nullable=false)
	private String lastName;

	@Column(name="email", nullable=false, unique=true)
	private String email;

	@Column(name="username", nullable=false, unique=true)
	private String username;

	@Column(name="password", nullable=false)
	private String password;

	//We can also setup multiplicity, do that we use annotataions such as @ManyToMany, @ManyToOne, @OneToMany
	//JoinColumn, and others

	//One user has many posts
	@OneToMany(mappedBy="user", cascade=CascadeType.ALL)
	//Json ignore will ignore the user object inside of the posts when we try to return data to the user
	@JsonIgnore
	private List<Post> posts;

	//Many to Many example, just so you can see one
	@ManyToMany(mappedBy="likes")
	@JsonIgnore
	private Set<Post> likePosts = new HashSet<Post>();

```

Spring Data JPA: Hierarchy

REVATURE

Basic CRUD functionality is provided by the JpaRepository interface

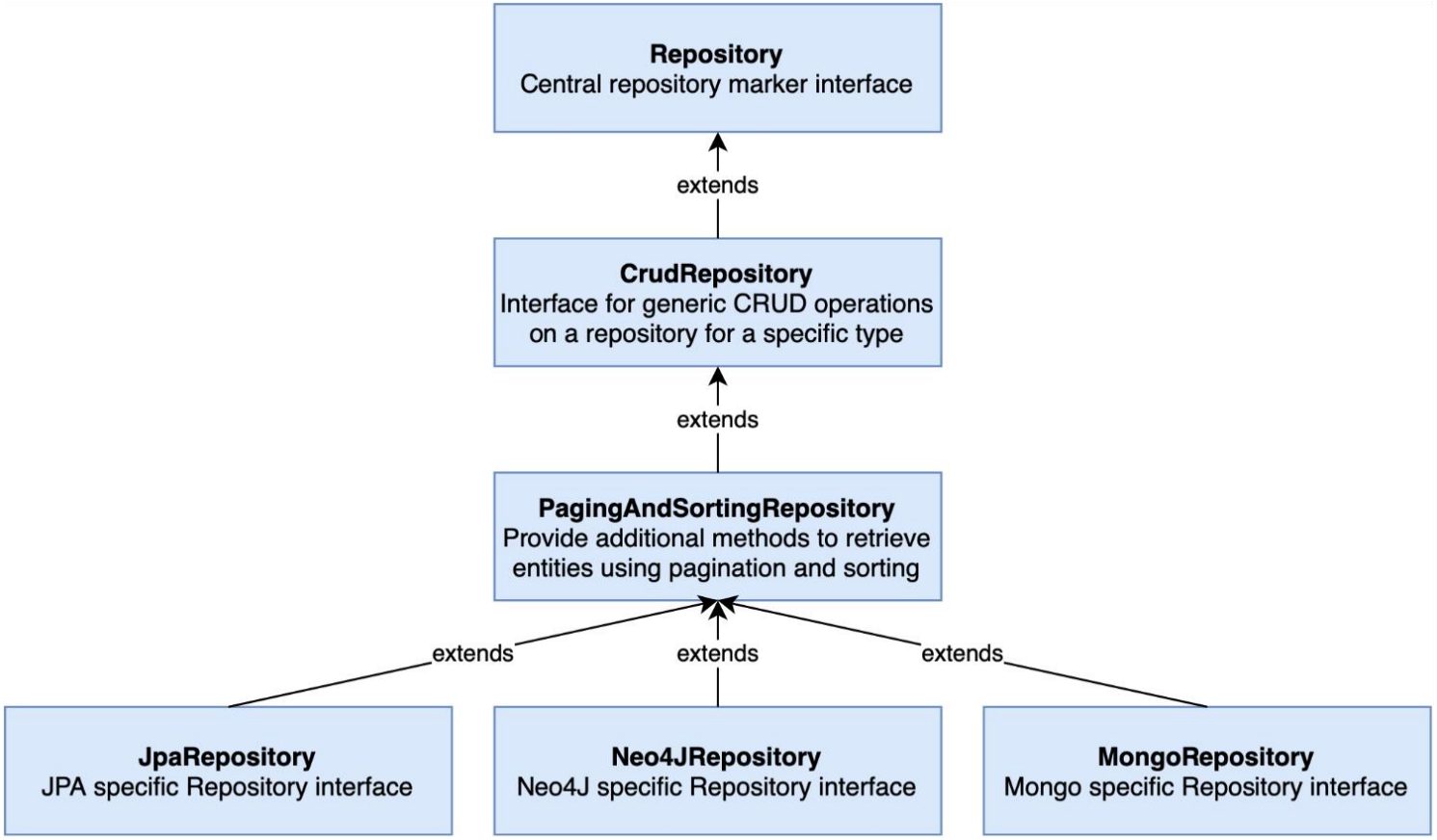

Image source: https://livebook.manning.com/book/cloud-native-spring-in-action/chapter-5/v-3/

Spring Data JPA: Features

REVATURE

- Building sophisticated repositories
- Support for QueryDSL
- Transparent Auditing of domain classes
- Pagination support
- Dynamic Query Execution
- Support for integration of custom data access code
- Automatic custom queries
- Validation of @Query annotated queries
- Support for XML based entity mapping
- JavaConfig based repositories

Spring Data JPA: Querying

REVATURE

There are three main ways to query data

- Predefined CRUD Methods
  - `https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html`
- Custom queries via method signature
  - `Table 2.3 > https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/jpa.repositories.html`
- Custom queries with @Query Annotation

Spring Data JPA: Annotations

REVATURE

Annotation	Purpose
@Transactional	Configure how the database transaction behaves
@NoRepositoryBean	Creates an interface that provides common methods for child repositories
@Param	Parameters can be passed to queries defined with @Query
@Transient	Mark a field as transient, to be ignored by the data store engine during reads and writes
@CreatedBy, @LastModifiedBy	Auditing annotations that will automatically be filled with the current principal
@CreatedDate, @LastModifiedDate	Auditing annotations that will automatically fill with current date
@Query	Supply a JPQL query for repository methods

Spring Data JPA: @Transactional

REVATURE

Allows Spring to manage our database transactions

- Best practice to use @Transactional over your Service classes
- Springs treats the Classes/Methods marked with @Transactional as separate transactions

Attribute	Purpose
isolation	The transaction isolation level.
noRollbackFor	Defines zero (0) or more exception Classes, which must be subclasses of Throwable, indicating which exception types must not cause a transaction rollback.
noRollbackForClassName	Defines zero (0) or more exception names (for exceptions which must be a subclass of Throwable) indicating which exception types must not cause a transaction rollback.
propagation	The transaction propagation type.
readOnly	A boolean flag that can be set to true if the transaction is effectively read-only, allowing for corresponding optimizations at runtime.
rollbackFor	Defines zero (0) or more exception classes, which must be subclasses of Throwable, indicating which exception types must cause a transaction rollback.
rollbackForClassName	Defines zero (0) or more exception names (for exceptions which must be a subclass of Throwable), indicating which exception types must cause a transaction rollback.
timeout	The timeout for this transaction (in seconds).
transactionManager	A qualifier value for the specified transaction.

@Transactional Propagation Strategy

REVATURE

- Defines how Spring should manage multi-level queries
  - One service class calling another

Strategy	Purpose
Mandatory	Support a current transaction, throw an exception if none exists.
Nested	Execute within a nested transaction if a current transaction exists, behave like REQUIRED otherwise.
Never	Execute non-transactionally, throw an exception if a transaction exists.
Not Supported	Execute non-transactionally, suspend the current transaction if one exists.
Required	Support a current transaction, create a new one if none exists.
Requires New	Create a new transaction, and suspend the current transaction if one exists.
Supports	Support a current transaction, execute non-transactionally if none exists.

Properties of a Transaction

REVATURE

Every transaction should abide by the ACID properties

- Atomic: All or nothing
- Consistency: Schema stays intact after each transaction
- Isolation: Transactions don’t interfere with other transactions
- Durability: Data persists through issues