Being productive with JPA using

Spring Data, DeltaSpike Data & QueryDSL

Matti Tahvonen, Developer advocate @ Vaadin
@mstahv, github.com/mstahv

vaadin }>

Who am I and why would I know?

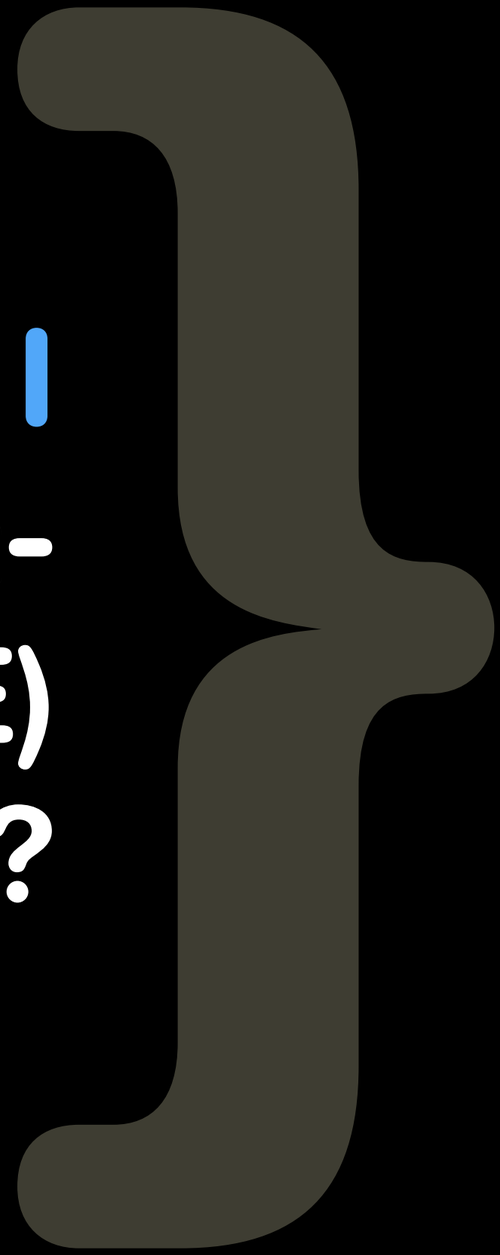
- Not really at my comfort zone with JPA :-)
- My job is awesome!
- I’m not here to sell these tools, just a big fan!

Matti Tahvonen, Developer advocate @ Vaadin
@mstahv, github.com/mstahv

Agenda

Part I

Why would I use non-standard (non Java EE) libraries?

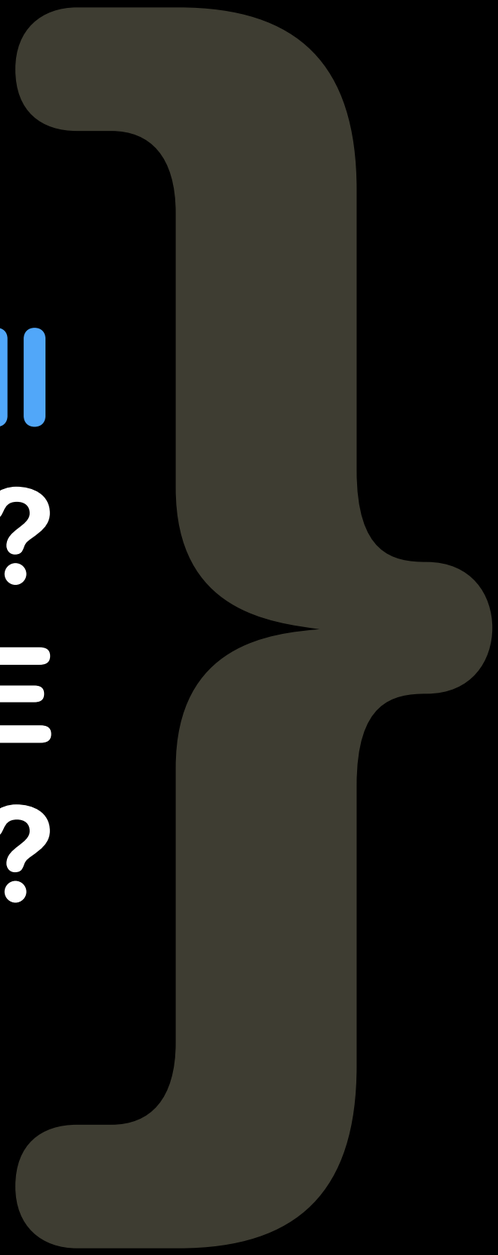


vaadin }>

Agenda

Part II

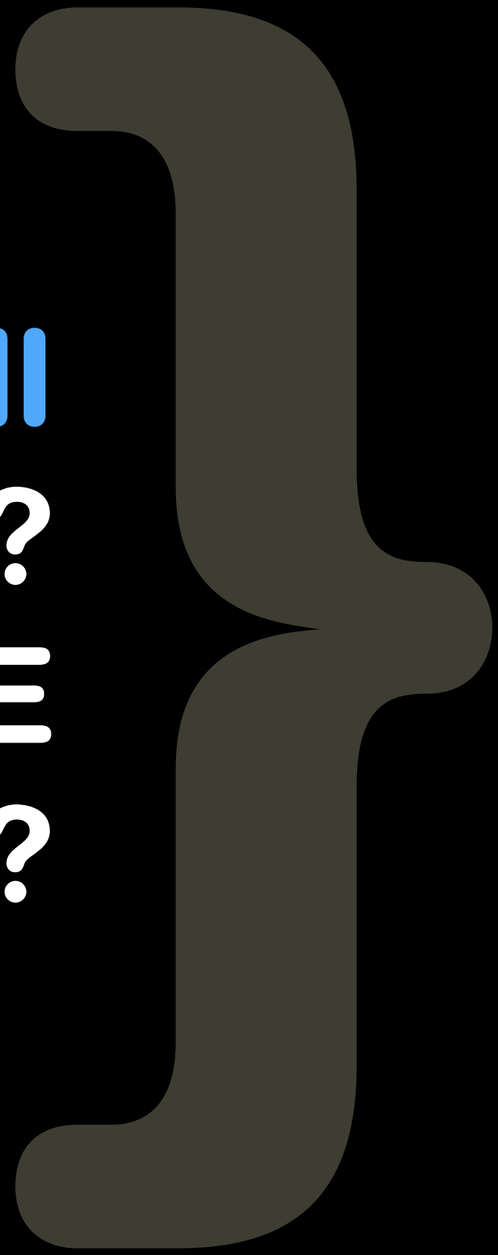
DeltaSpike Data?
Usage in Java EE environment?



Agenda

Part III

Query DSL?
Usage in Java EE environment?

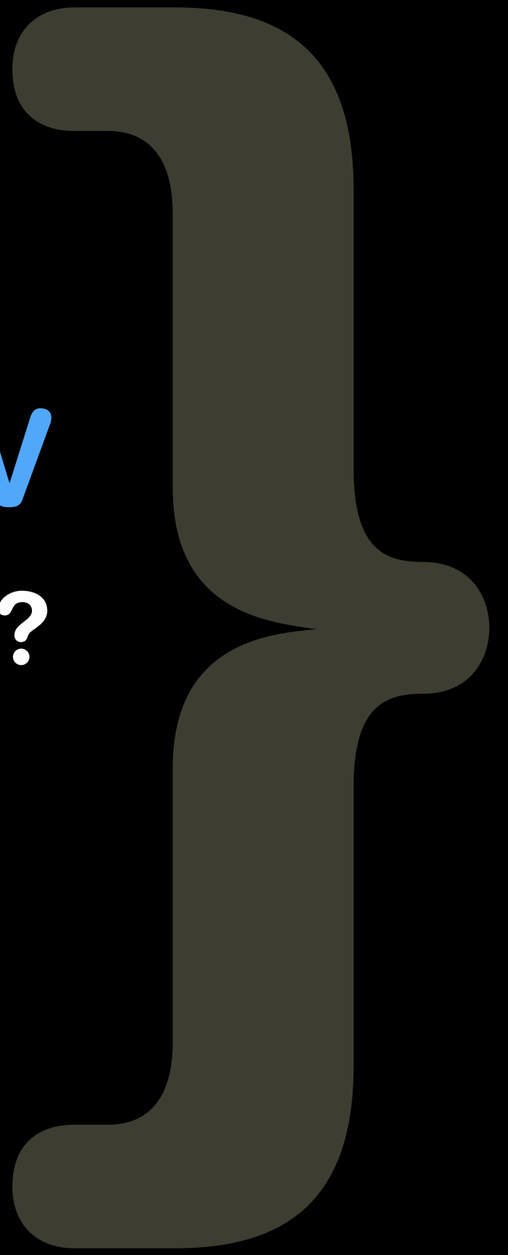


vaadin }>

Agenda

Summary IV

Which helper library to use?



vaadin }>

JPA

JPA/ORM haters

-> mostly due wrong expectations

-> it is a leaky abstraction, accept that!

vaadin }>

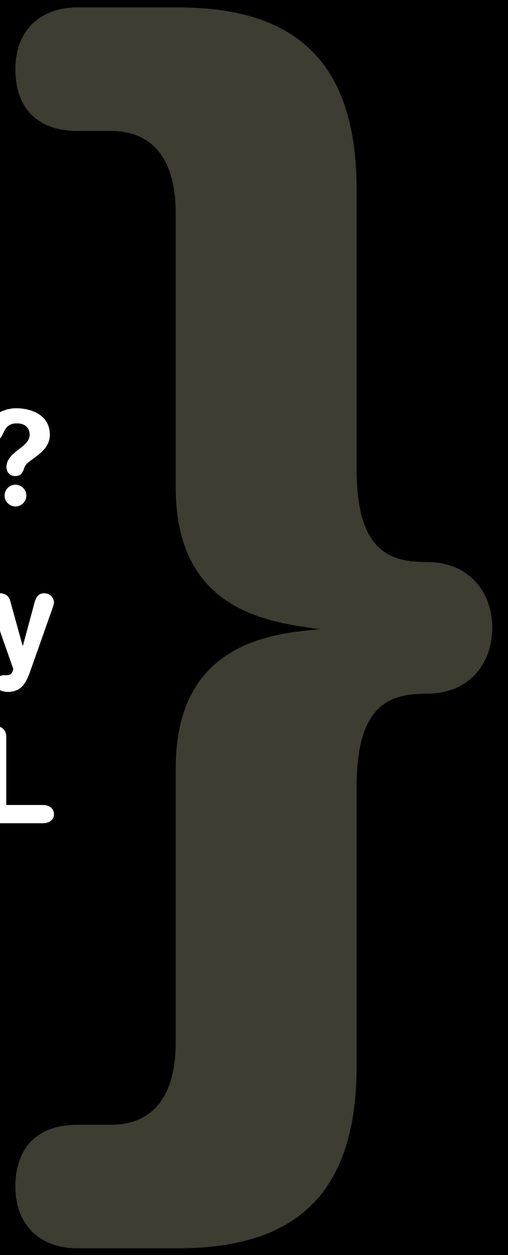
JPA

Mapping part is already quite complete in 2.0/2.1

vaadin }>

JPA

Query part?
JPQL is still a query language ~ SQL



vaadin }>

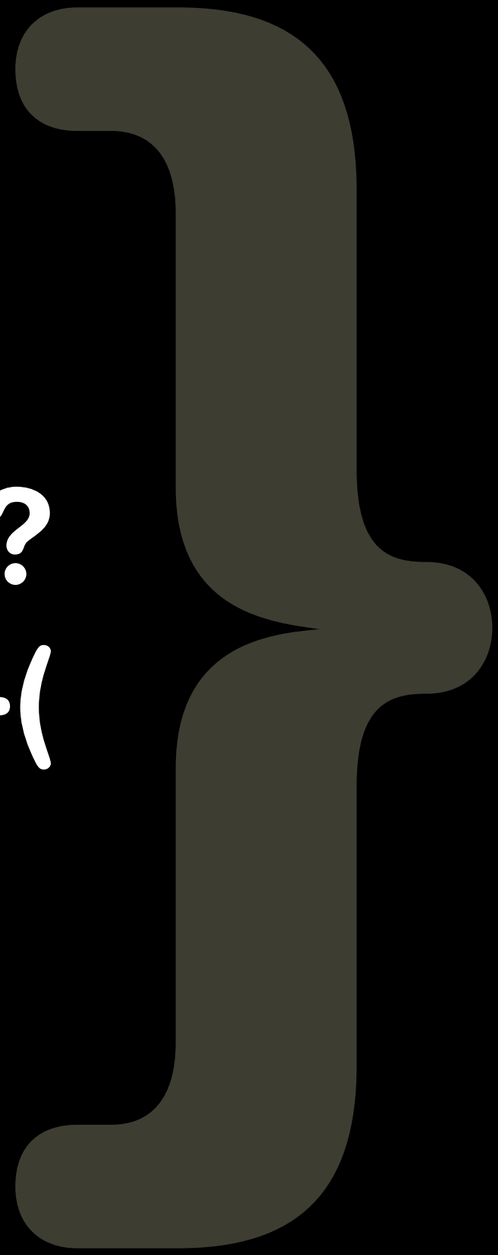
JPA

Query part?
Criteria API just makes it
more complex

vaadin

JPA

Query part?
No query-by-example :-(



vaadin }>

JPA

Query part?

Building DAOs with plain JPA:

Error prone

Non-productive

You'll repeat yourself

I used to avoid JPA in my Vaadin usage examples!

JPA

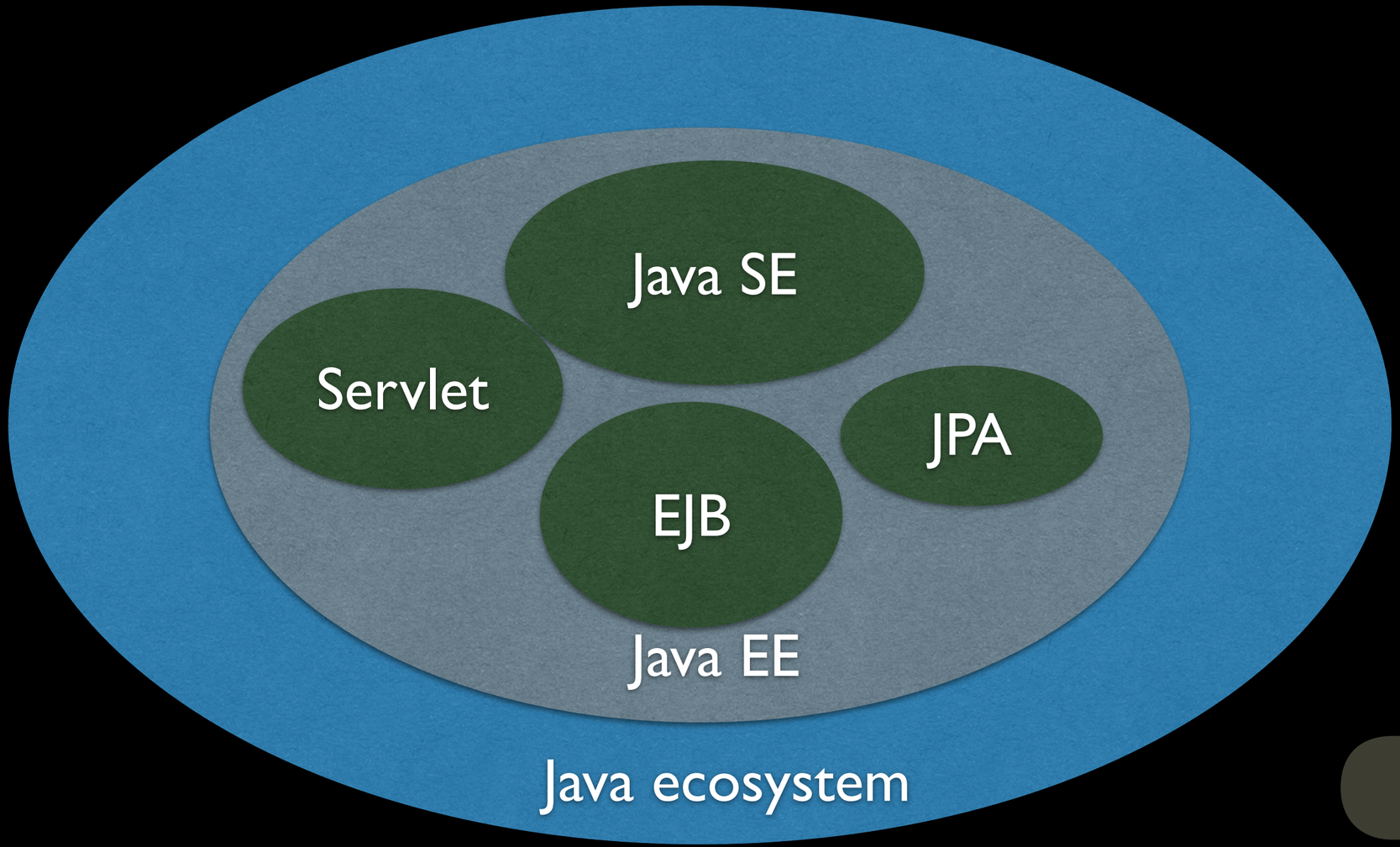
"Problems" with standards

-> hard to cover everything

-> moves slowly

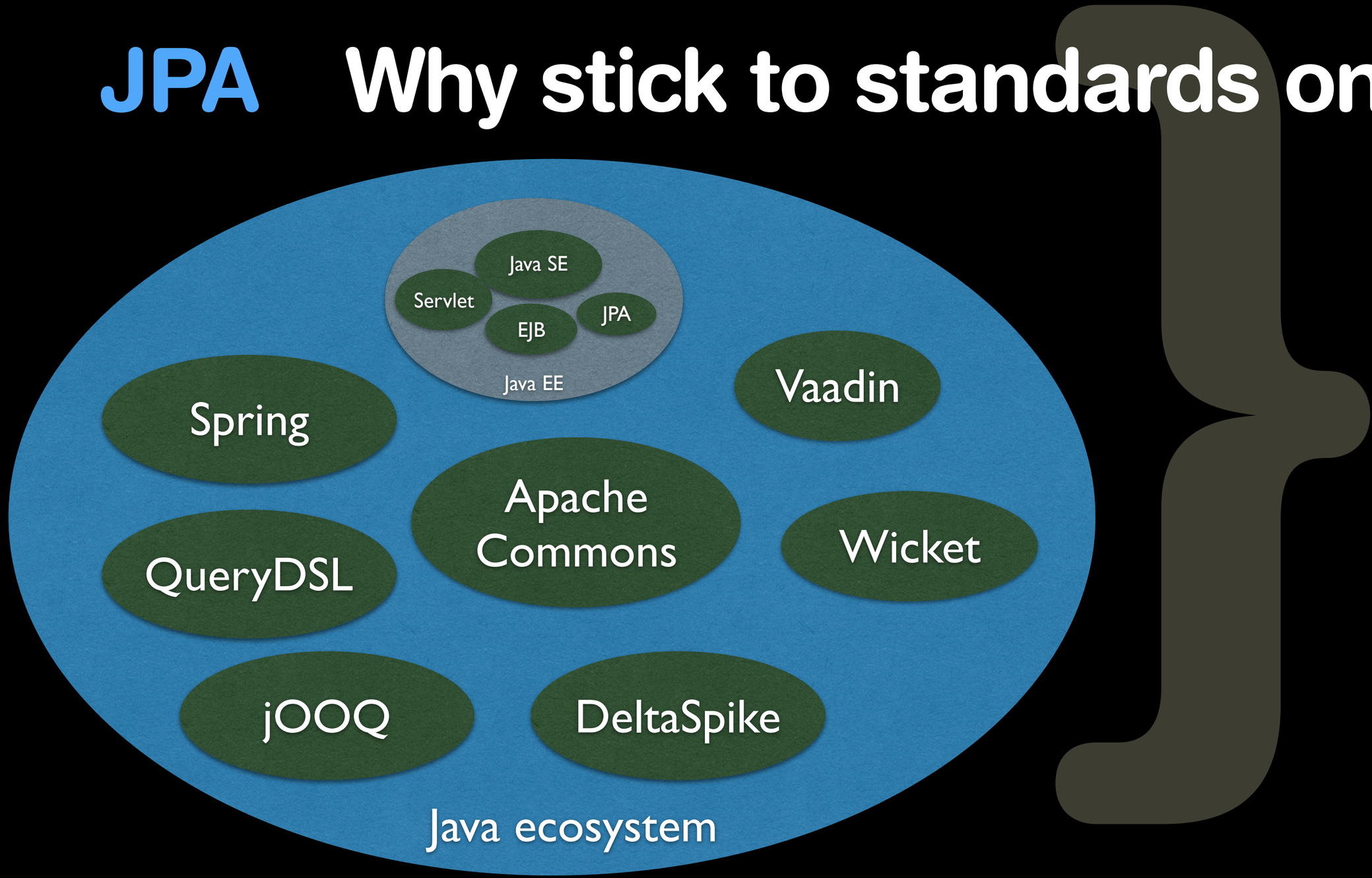
-> compromises

JPA Java EE purist?



vaadin }>

JPA Why stick to standards only?



vaadin }>

JPA

"Spring Data made me want to use JPA in my Vaadin examples."

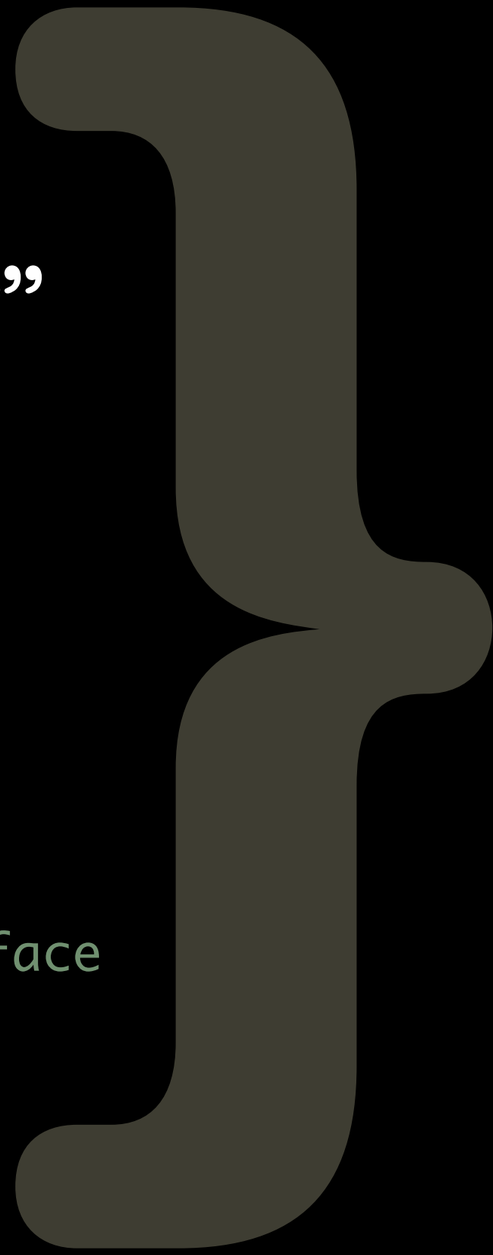
vaadin }>

JPA

Spring Data JPA “repository library”

- Simple things are dead simple
- Avoid sh!tload of boilerplate code
- Easy to fall back to lower level JPA API

```
public interface PersonRepository extends
        JpaRepository<Person, Long> {

    // save, delete, findAll etc from super interface

    List<Person> findByLastName(String lastName);
}
```

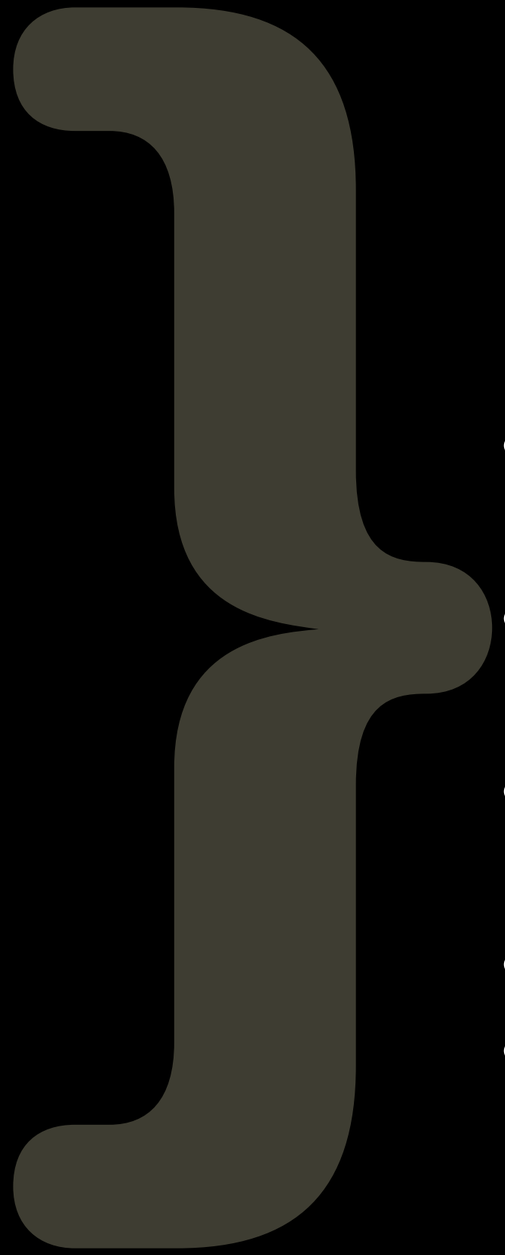


vaadin }>

DeltaSpike Data

a Spring Data “clone” in plain CDI

DeltaSpike Data



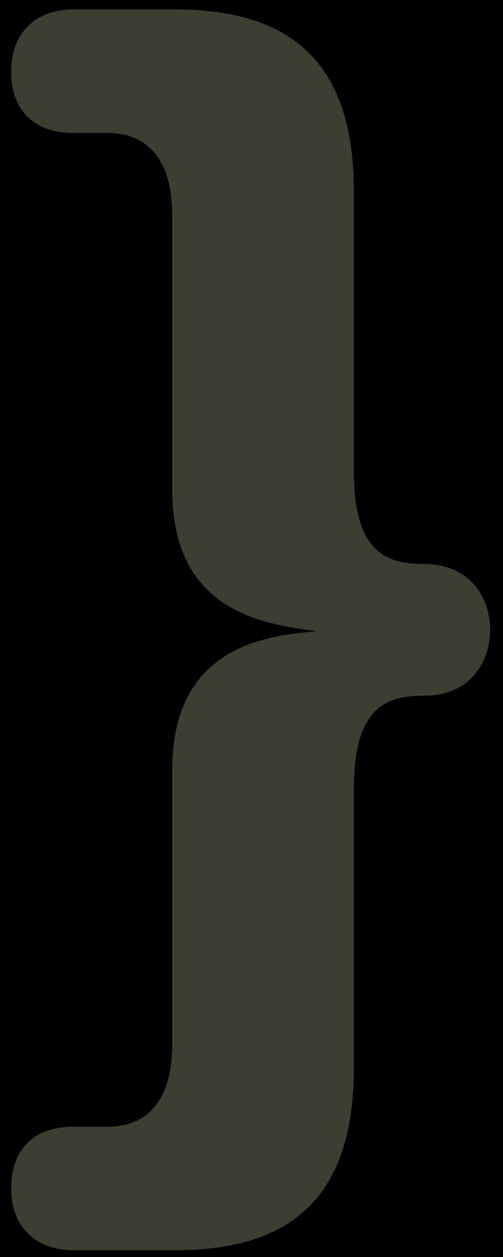
Repository principles

- 1 repository per entity (interface/abstract class)
- Container generates the implementation with methods suitable for CRUD
- Simple custom queries by method naming convention only!
- Easy to “escape” to JPQL when needed
- Typically no need to access EntityManager

DeltaSpike Data

Example repository

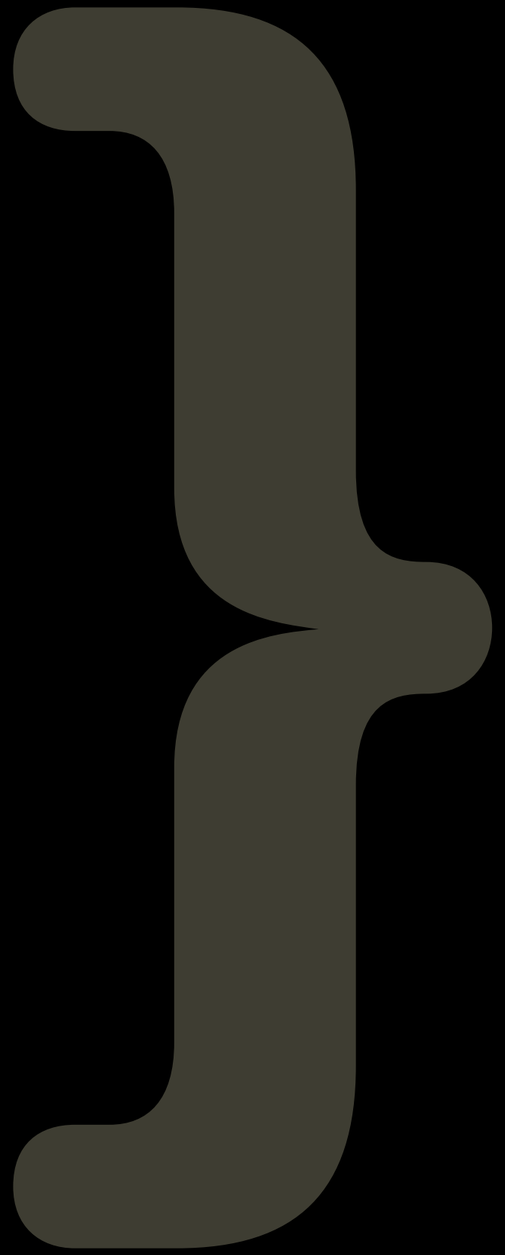
```
@Repository(forEntity = User.class)
public interface UserRepository extends
        EntityRepository<User, Long> {

    public User findByEmail(String email);

}
```



DeltaSpike Data

Java EE setup

- Could be used with plain CDI, @Transactional and with user managed persistency context and transactions
- Java EE evangelists like Arun Gupta, David Blevins, Adam Bean would rip their pants!
- Modern JPA usage mantras for Java EE: container provided persistency context, JTA transactions, super simple EJBs…

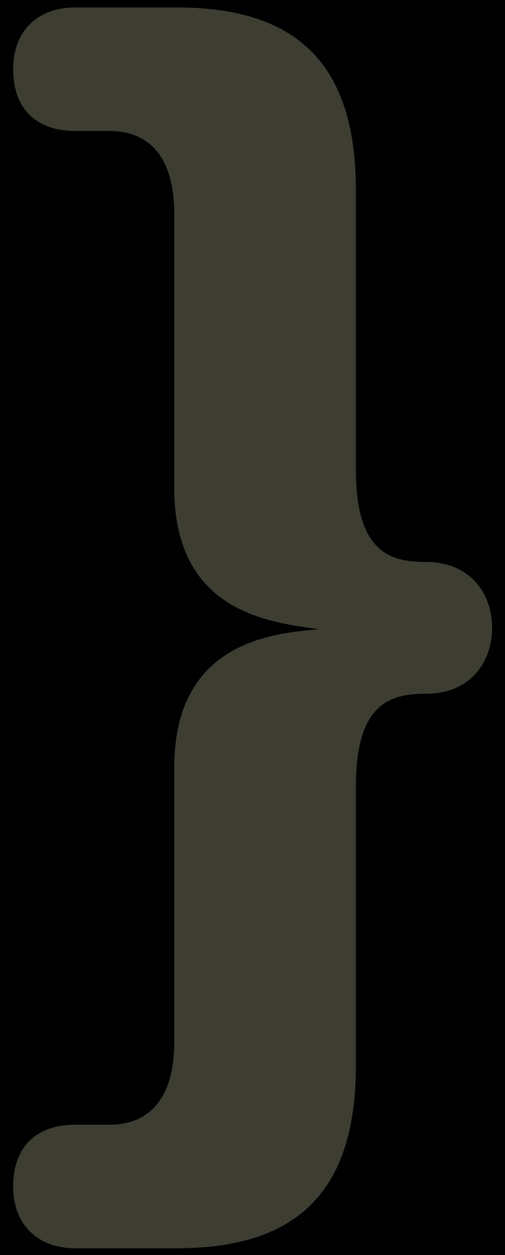


DeltaSpike Data

Java EE setup, a dead simple recipe

- Inject repository to @Stateless EBJ and expose API from there
  - EJB handles transactions, pooling etc
  - Possible to limit/modify the API seen by UI layer
  - Access multiple repositories in same transaction
  - Really easy to use low level JPA API for some methods when needed

vaadin }>

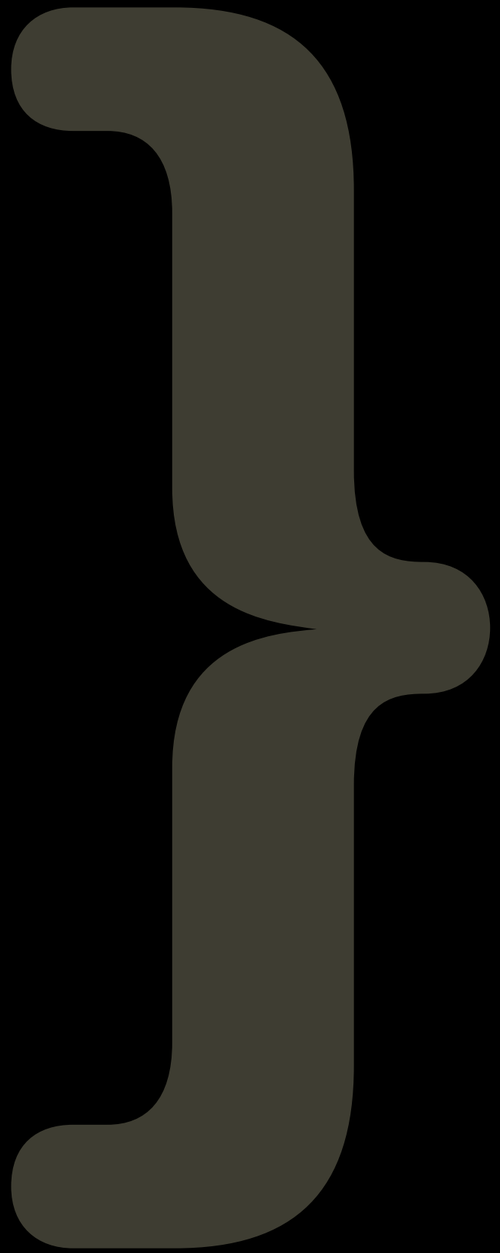


DeltaSpike Data

Let’s code!

- A Java EE app stub without backend (JPA preconfigured)
- Tasks:
  1. Add dependencies
  2. Expose persistency context as CDI bean
  3. Disable transaction management by DeltaSpike Data (EJB will handle this)
  4. Introduce a repository
  5. Introduce an EJB in front

vaadin }>



DeltaSpike Data

1. Dependencies

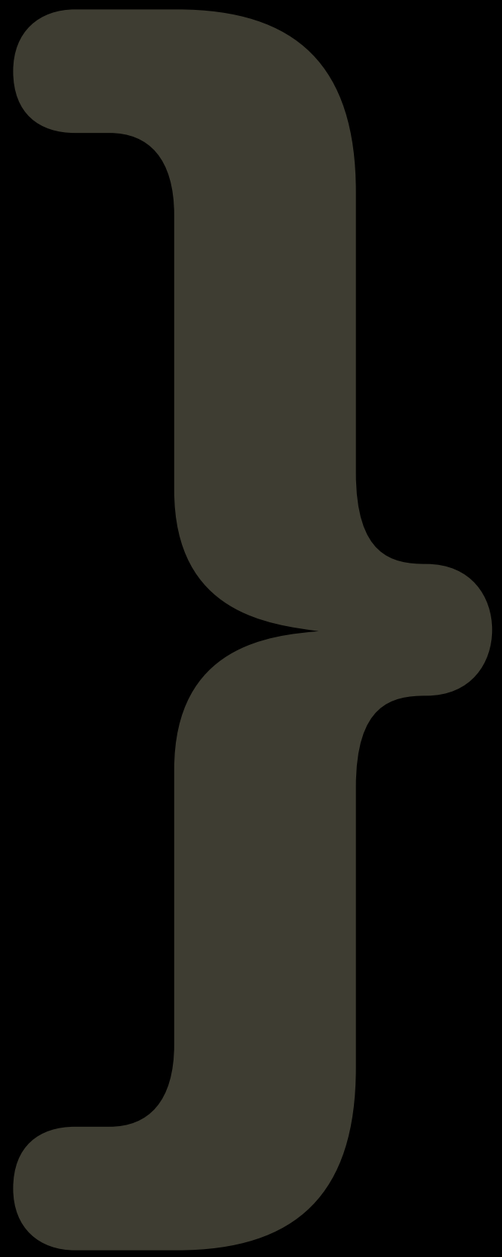
DeltaSpike Core +

```
<dependency>
    <groupId>org.apache.deltaspike.modules</groupId>
    <artifactId>deltaspike-data-module-api</artifactId>
    <version>1.5.0</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.apache.deltaspike.modules</groupId>
    <artifactId>deltaspike-data-module-impl</artifactId>
    <version>1.5.0</version>
    <scope>runtime</scope>
</dependency>
```

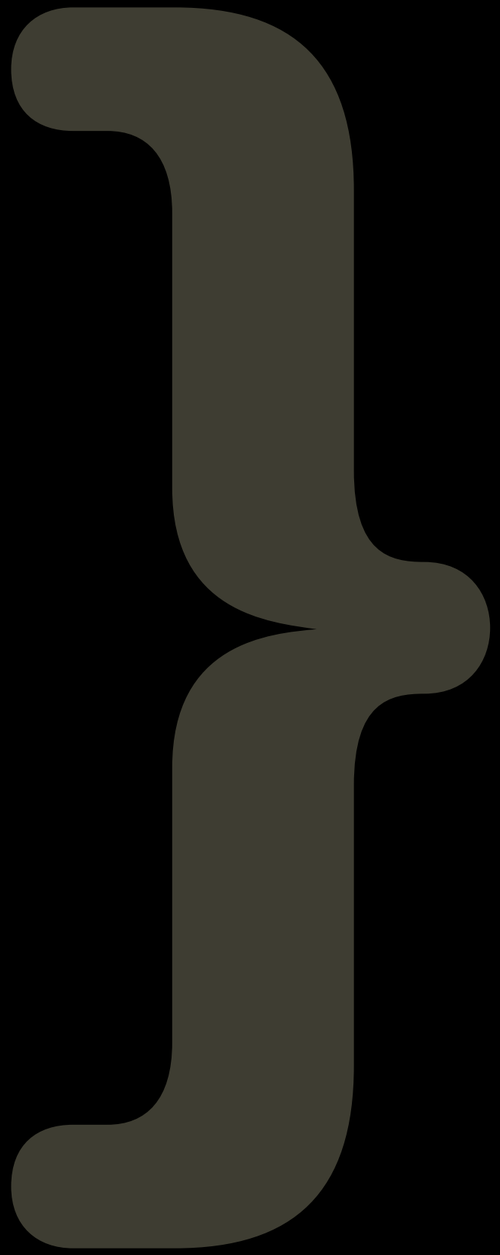
DeltaSpike Data

2. Expose persistency context as CDI bean

```
public class CdiConfig {

    @Produces
    @Dependent
    @PersistenceContext(unitName = “your-pu-name“)
    public EntityManager entityManager;

}
```



vaadin }>



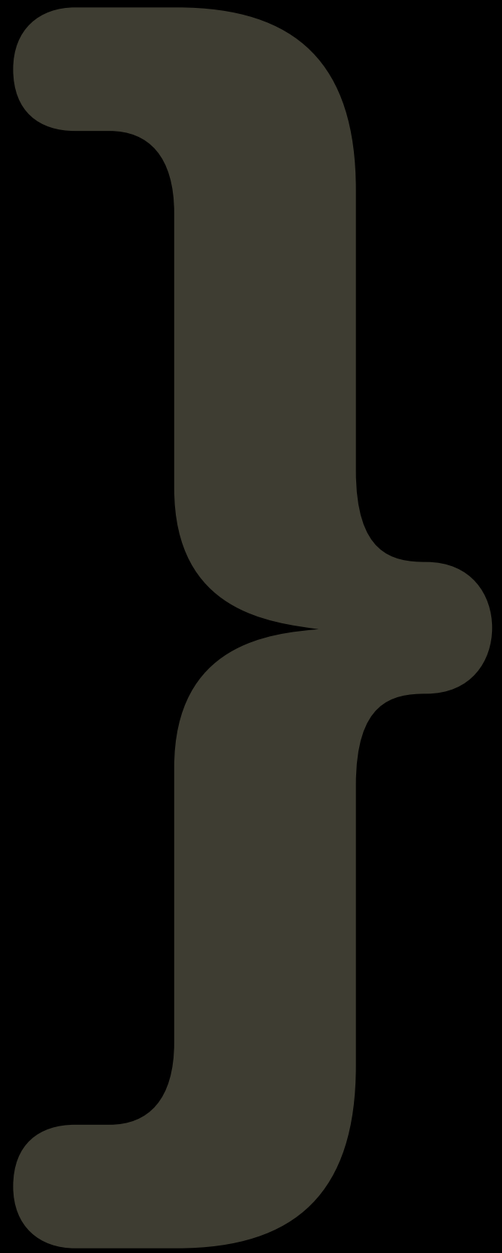
DeltaSpike Data

3. Disable transaction management by DeltaSpike Data

META-INF/apache-deltaspike.properties:

```
globalAlternatives.org.apache.deltaspike.jpa.spi.transaction.TransactionStrategy=org.apache.deltaspike.jpa.impl.transaction.ContainerManagedTransactionStrategy
```

vaadin }>



DeltaSpike Data

4. Repository

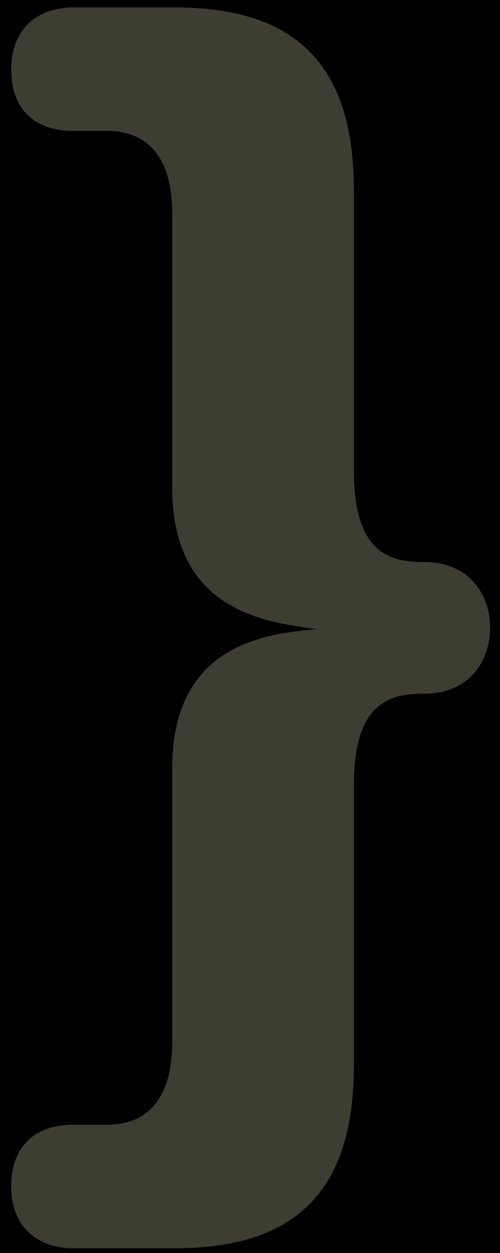
```
@Repository
public interface BookRepository extends
        EntityRepository<Book, Long> {

    public List<Book> findByCategory(Category value);

}
```

vaadin }>

DeltaSpike Data

5. Service class (EJB)

```
@Stateless
public class LibraryFacade {

    @Inject
    private BookRepository bookRepository;

    public void save(Book value) {
        bookRepository.save(value);
    }

    public List<Book> findAll() {
        return bookRepository.findAll();
    }
    //...
}
```



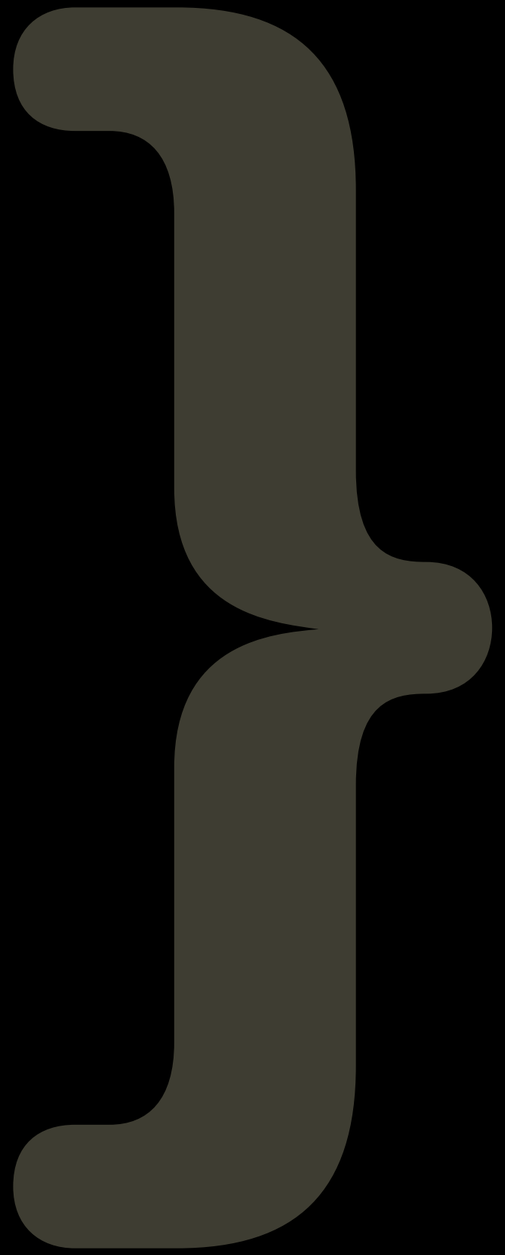
vaadin }>

DeltaSpike Data

Paging

```
@Repository
public interface BookRepository extends
        EntityRepository<Book, Long> {

    public List<Book> findByCategory(Category value,
                    @FirstResult int start,
                    @MaxResults int pageSize);
}
```

vaadin }>



DeltaSpike Data

Paging, using QueryResult

- “Intermediate query result”, that can handle e.g. paging & sorting

```
public List<Book> findBooksByCategory(Category value, int first,
        int maxresults, String sortProperty) {
    QueryResult<Book> qr = bookRepository.findByCategory(value);
    qr.firstResult(first)
            .maxResults(maxresults)
            .orderAsc(sortProperty);
    return qr.getResultList();
}
```

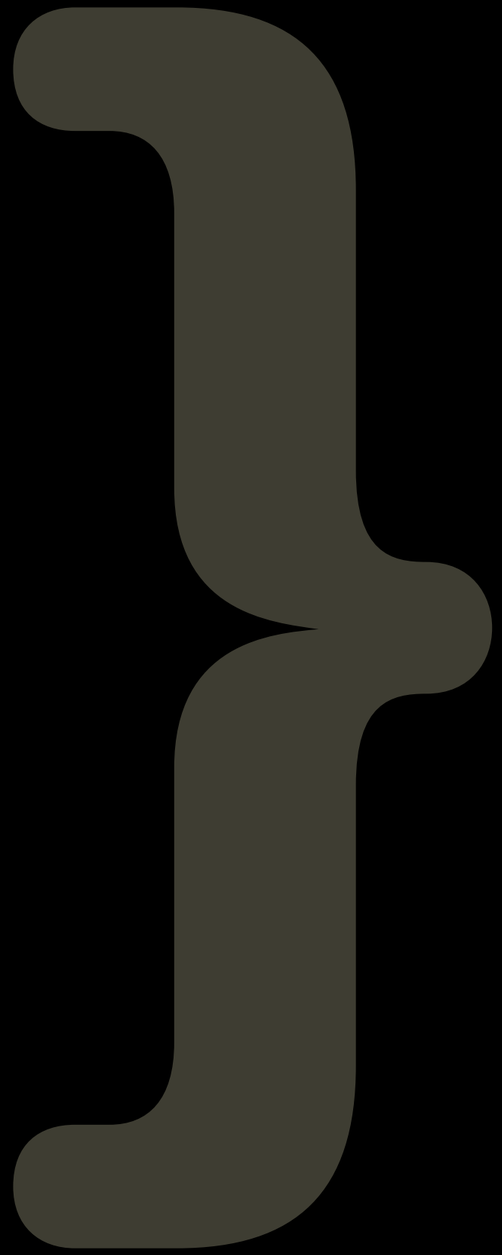
vaadin }>

DeltaSpike Data

Adding methods using naming convention

```
@Repository
public interface BookRepository extends
        EntityRepository<Book, Long> {

    public List<Book> findByCategory(Category value);

}
```

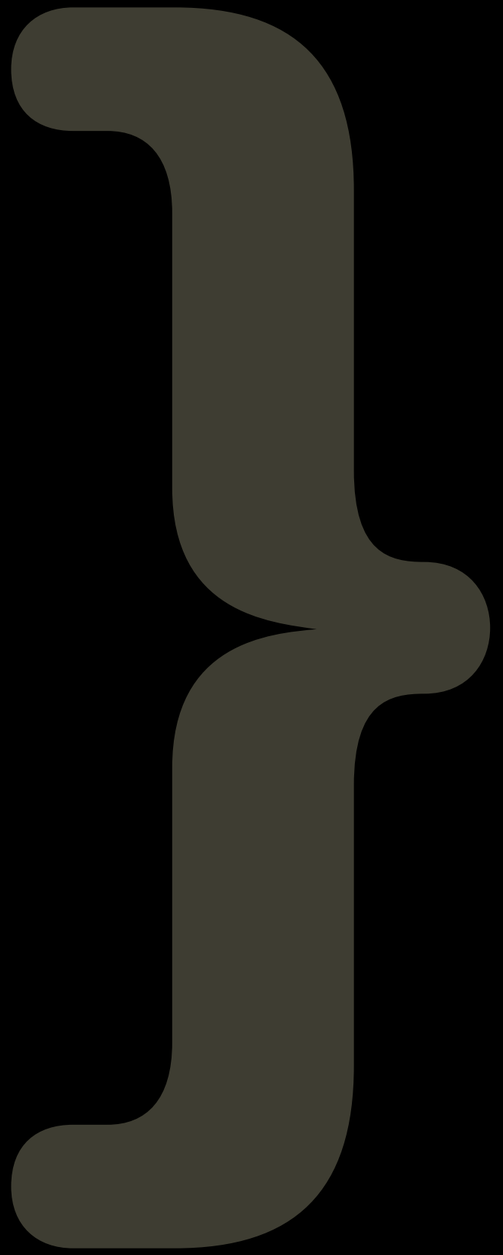
vaadin }>



DeltaSpike Data

The problem with method name magic: too long method names

```
public List<Workout>
findByPersonAndWorkoutDateGreaterThanAndWorkoutDateLessThanAnd
DescriptionLikeIgnoreCase(
            Person currentUser, Date beginDate, Date endDate,
String filter);
```



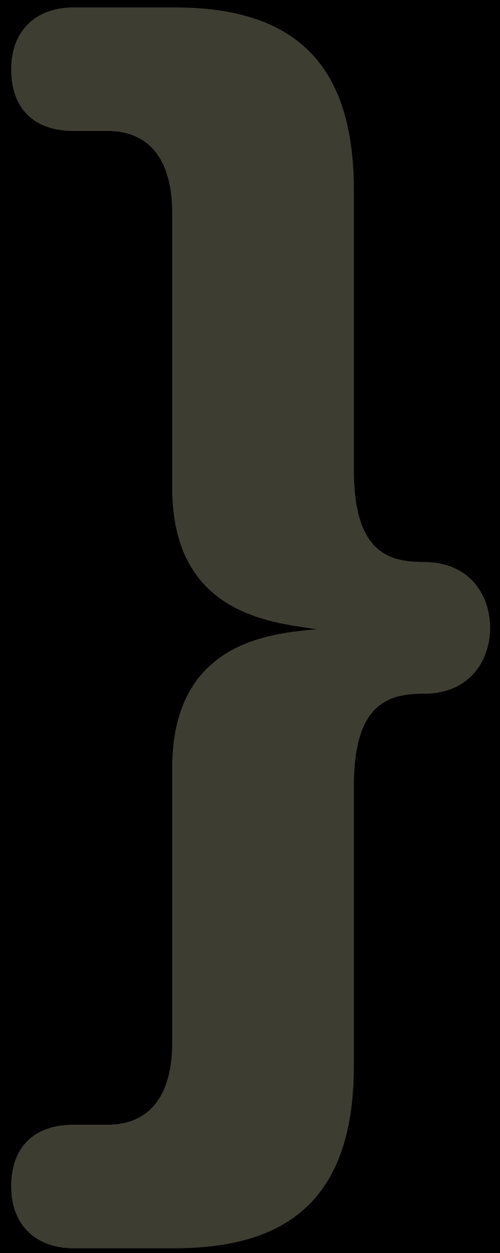
DeltaSpike Data

Alternative ways to define queries

- JPQL/SQL with @Query annotation
- Named queries

```
@Repository
public interface BookRepository extends EntityRepository<Book, Long> {

    public List<Book> findByCategory(Category value);

    @Query("SELECT FROM Person p WHERE category = ?1")
    public List<Book> findByCategoryV2(Category value);

    @Query(named = "Person.byCategory")
    public List<Book> findByCategoryV4(Category value);

}
```

vaadin }>



DeltaSpike Data

Alternative ways to define queries

- JPA library agnostic query by example Implementation !

```
public List<Book> findBooks(String filter) {
    Book example = new Book();
    example.setName(filter);
    return bookRepository.findByLike(example, Book_.name);
}
```

vaadin }>

DeltaSpike Data

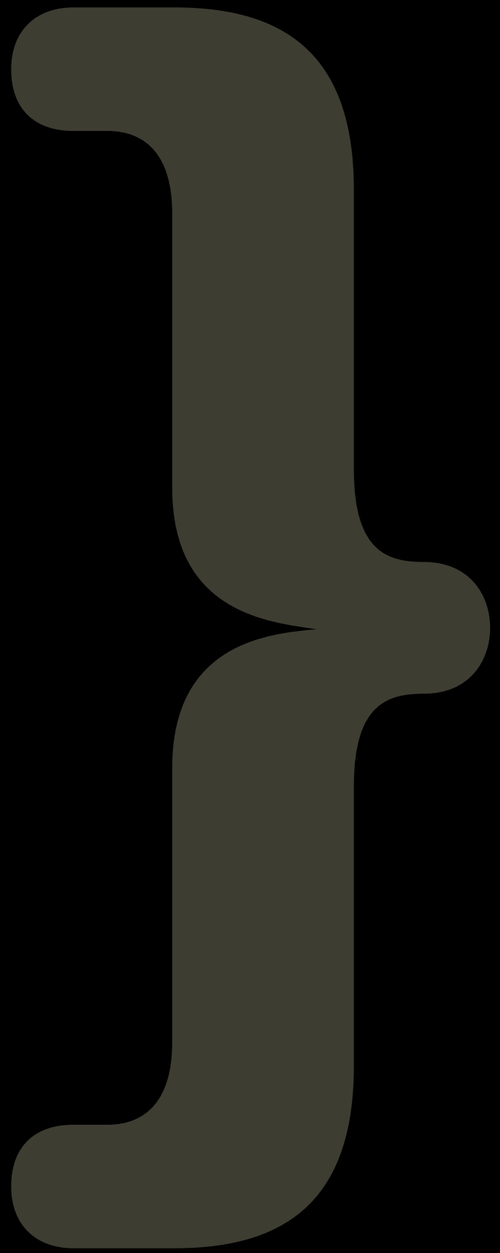
Alternative ways to define queries

- Criteria API helpers in DeltaSpike Data

```
@Repository(forEntity = Person.class)
public abstract class PersonRepository implements CriteriaSupport<Person> {
   public List<Person> findAdultFamilyMembers(String name, Integer minAge) {
        return criteria()
                .like(Person_.name, "%" + name + "%")
                .gtOrEq(Person_.age, minAge)
                .eq(Person_.validated, Boolean.TRUE)
                .orderDesc(Person_.age)
                .getResultList();
    }
}
```

Example from : https://deltaspike.apache.org/documentation/data.html

vaadin }>



DeltaSpike Data

Extensions

Open for extension, e.g. documentation site has an example how to add QueryDSL support

vaadin }>

QueryDSL

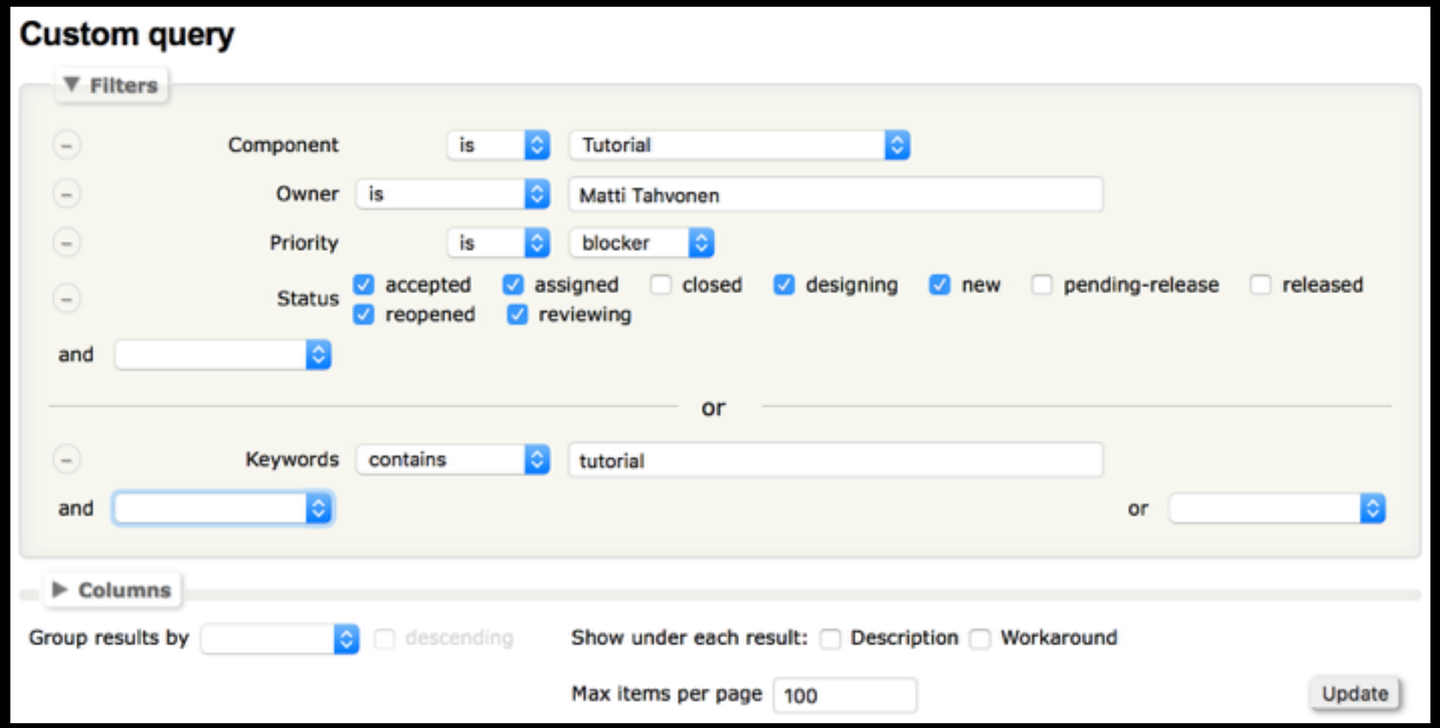
QueryDSL

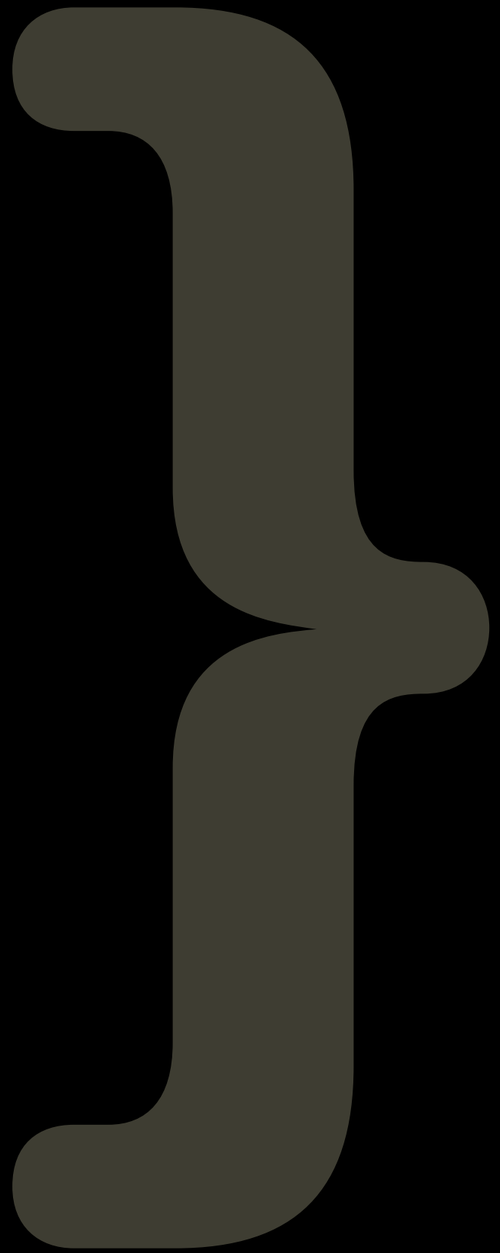
A compact type-safe Java API to
(dynamically) construct queries

vaadin }>

QueryDSL

Don’t try this with repositories & method naming conventions :-)





vaadin

QueryDSL

Programmatic* approach excels over static queries** in:

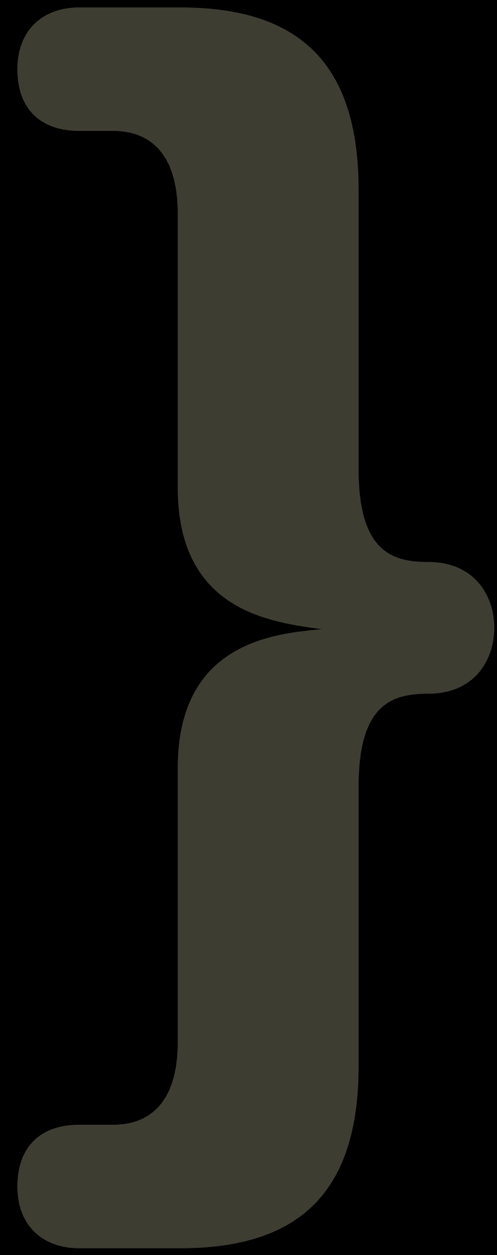
- Long queries
- Complex queries
- Dynamic queries

*) Criteria API, QueryDSL etc
*) Repositories methods, (JP/S)QL queries

QueryDSL

- Bit like Criteria API, but very different
- Bit like jOOQ, but for JPA
- (but not for JPA only)

```
QCustomer customer = QCustomer.customer;
JPAQuery query = new JPAQuery(entityManager);
Customer bob = query.from(customer)
   .where(customer.firstName.eq("Bob"))
   .uniqueResult(customer);
```

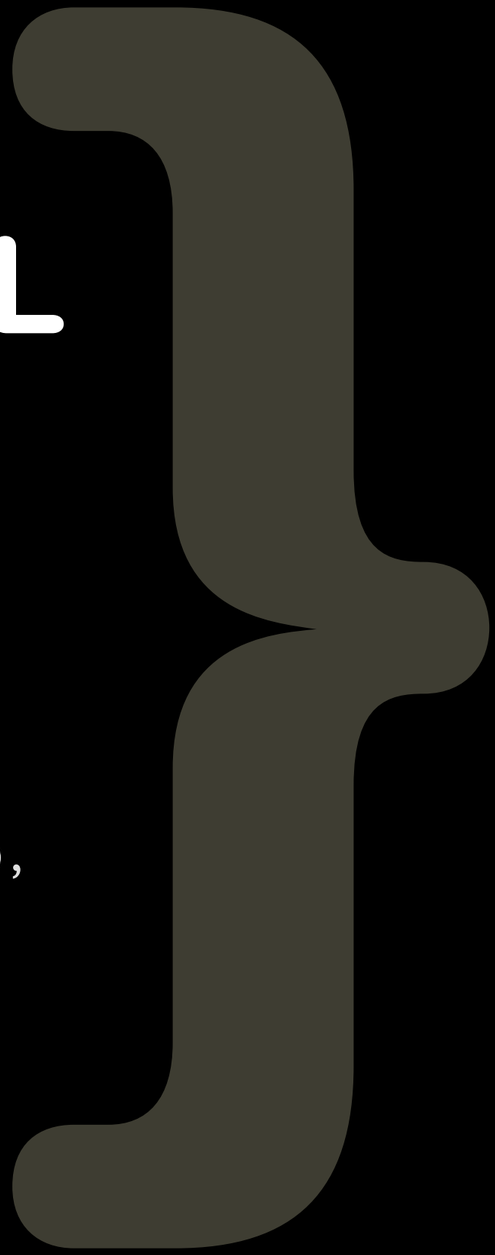


vaadin }>

QueryDSL

Criteria API vs QueryDSL

```
CriteriaQuery query = builder.createQuery();
Root<Person> men = query.from( Person.class );
Root<Person> women = query.from( Person.class );
Predicate menRestriction = builder.and(
    builder.equal( men.get( Person_.gender ), Gender.MALE ),
    builder.equal( men.get( Person_.relationshipStatus ),
RelationshipStatus.SINGLE ));
Predicate womenRestriction = builder.and(
    builder.equal( women.get( Person_.gender ), Gender.FEMALE ),
    builder.equal( women.get( Person_.relationshipStatus ),
RelationshipStatus.SINGLE ));
query.where( builder.and( menRestriction, womenRestriction ) );
```



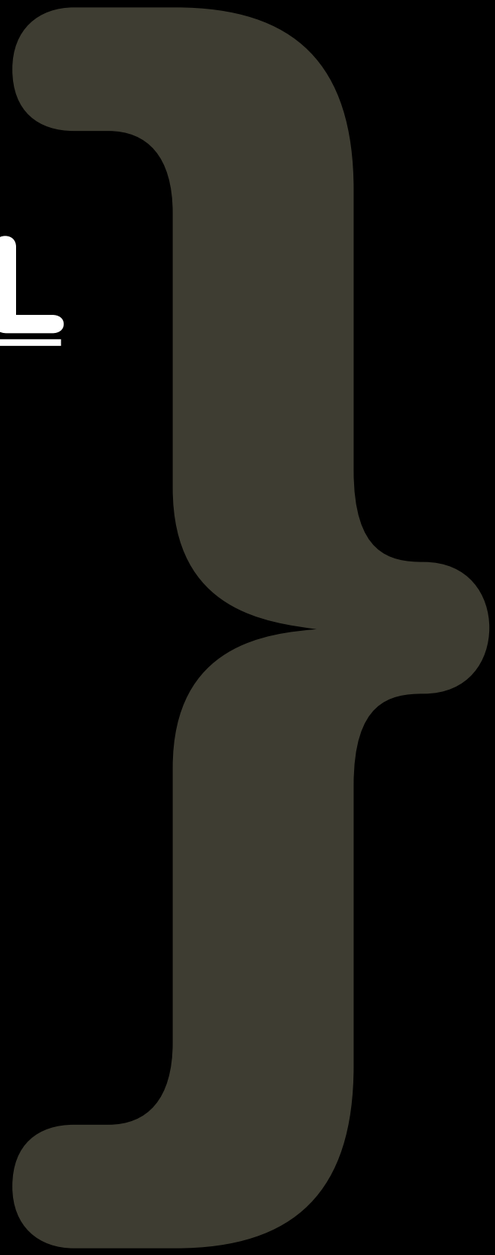
Example from : http://blog.mysema.com/2010/04/querydsl-as-alternative-to-jpa-2.html

vaadin }>

QueryDSL

Criteria API vs QueryDSL

```
JPAQuery query = new JPAQuery(em);
QPerson men = new QPerson("men");
QPerson women = new QPerson("women");
query.from(men, women).where(
      men.gender.eq(Gender.MALE),
      men.relationshipStatus.eq(RelationshipStatus.SINGLE),
      women.gender.eq(Gender.FEMALE),
      women.relationshipStatus.eq(RelationshipStatus.SINGLE));
```

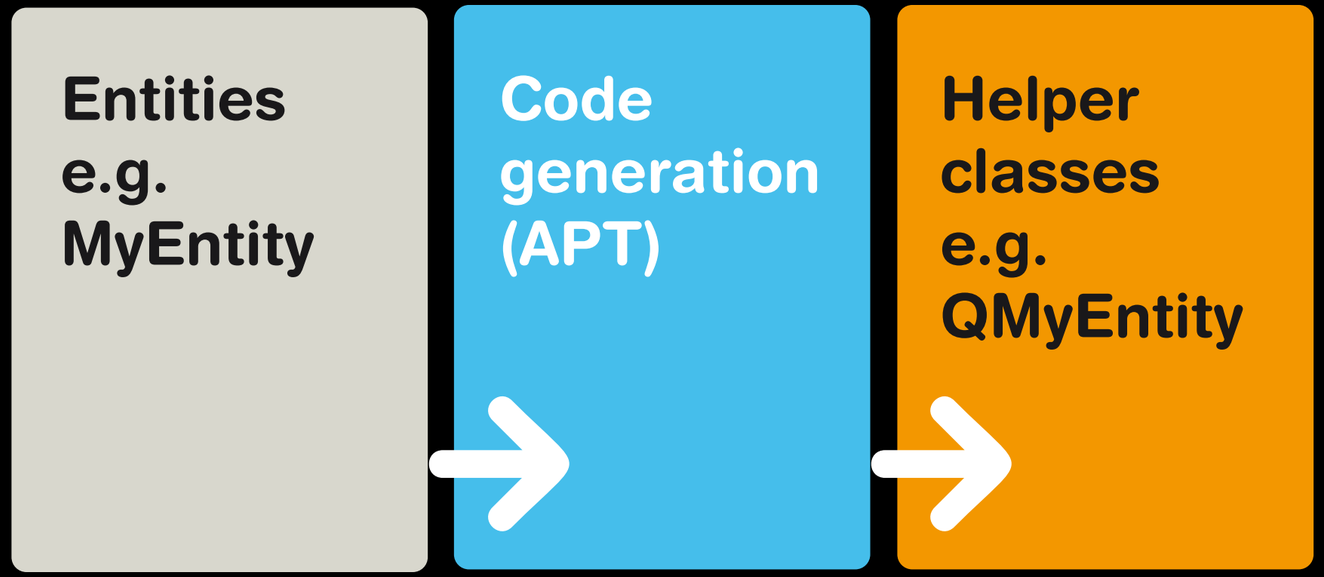


Example from : http://blog.mysema.com/2010/04/querydsl-as-alternative-to-jpa-2.html

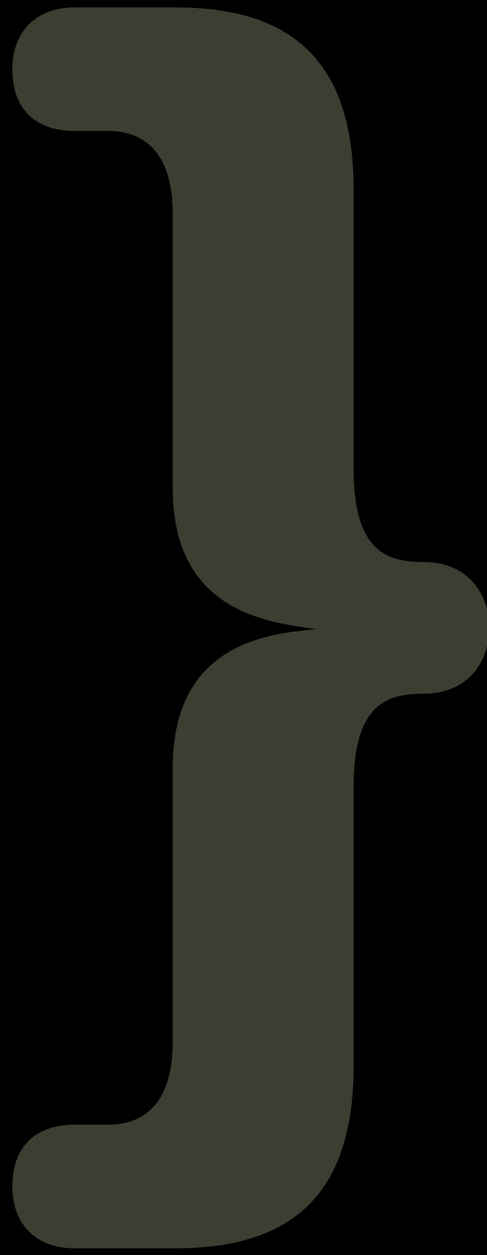
vaadin }>

QueryDSL

How does it work?



QueryDSL



Let’s code!

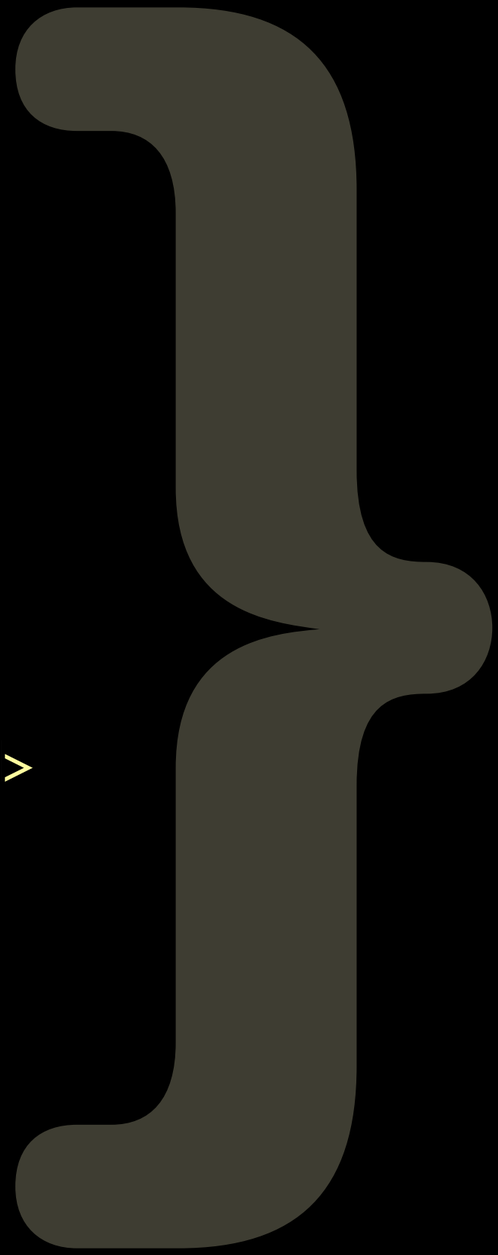
- Lets Configure QueryDSL to the example and use it for a query:
  1. Add dependencies
  2. Configure code generation
  3. Use it via EJB

vaadin }>

QueryDSL

1. Add dependencies

```
<dependency>
        <groupId>com.querydsl</groupId>
        <artifactId>querydsl-jpa</artifactId>
        <version>4.0.5</version>
</dependency>
```

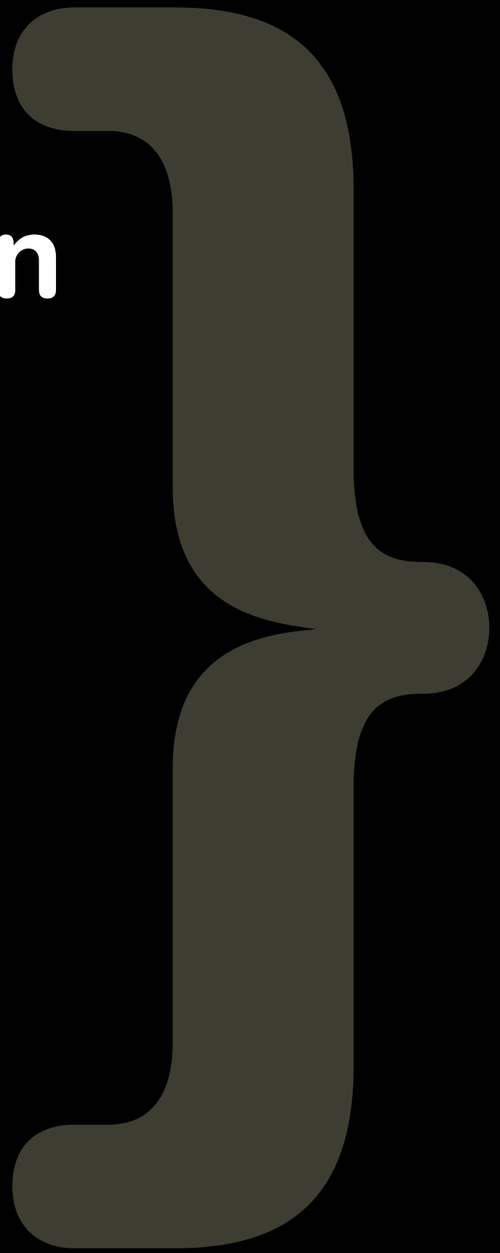


vaadin }>

QueryDSL

2. Configure code generation

```
<plugin>
  <groupId>com.mysema.maven</groupId>
  <artifactId>apt-maven-plugin</artifactId>
  <version>1.1.3</version>
  <executions>
    <execution>
      <id>querydsl</id>
      <goals><goal>process</goal></goals>
      <configuration>
        <outputDirectory>target/generated-sources/querydsl</outputDirectory>
        <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>com.querydsl</groupId>
      <artifactId>querydsl-apt</artifactId>
      <version>4.0.5</version>
    </dependency>
  </dependencies>
</plugin>
```

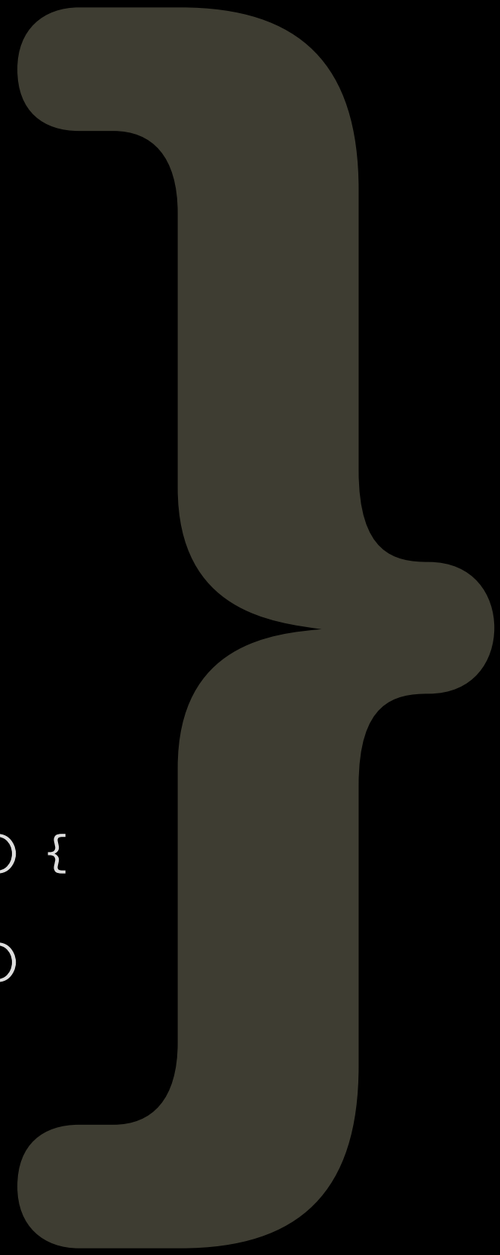


vaadin }>

QueryDSL

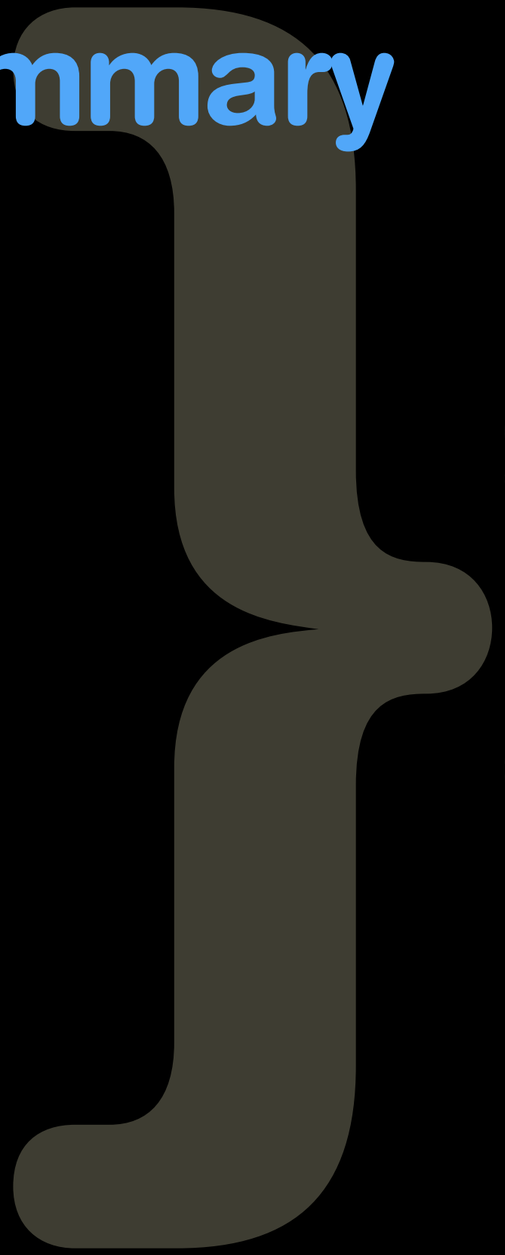
3. Use it via EJB

```
@PersistenceContext(name = "bookpu")
public EntityManager em;

private JPAQuery<Book> bookRoot() {
    return new JPAQuery<Book>(em).from(QBook.book);
}

public List<Book> findBooksByCategoryWithQueryDSL(Category cat) {
    return bookRoot()
            .innerJoin(QBook.book.category, QCategory.category)
            .where(QCategory.category.eq(cat))
            .fetch();
}
```



vaadin }>

Summary



Spring Data, DeltaSpike Data or QueryDSL?

vaadin }>

Summary

DeltaSpike Data or Spring Data and QueryDSL!

- Repository library for:
  - basic CRUD operations
  - simple static queries
  - DeltaSpike Data for fresh Java EE projects
- QueryDSL
  - when things get more complicated
- Hide the tools from you UI -> flexibility to refactor

Q&A

The example app:
https://github.com/mstahv/jpa-library-example

Matti Tahvonen, Developer advocate @ Vaadin
@mstahv, github.com/mstahv