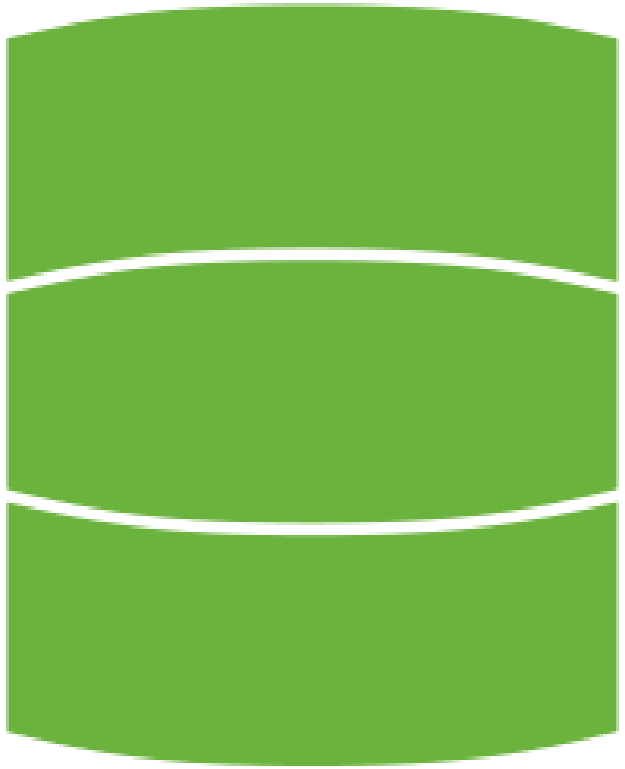


Spring Data

NOSERYOUNG

Spring Data

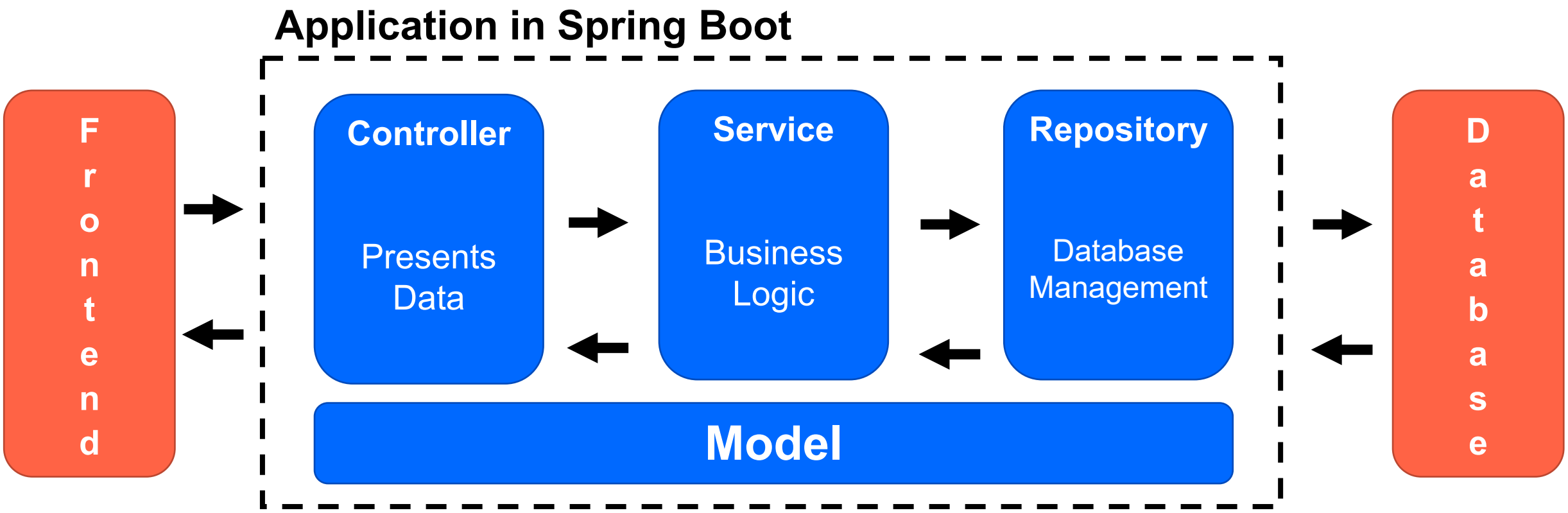
What is it?

- Spring Data is **a Spring module for managing data base access in Java applications**
- Spring Data allows programmers to write object-oriented code to interact with databases

NOSERYOUNG

Structure of a Spring Project

Repetition

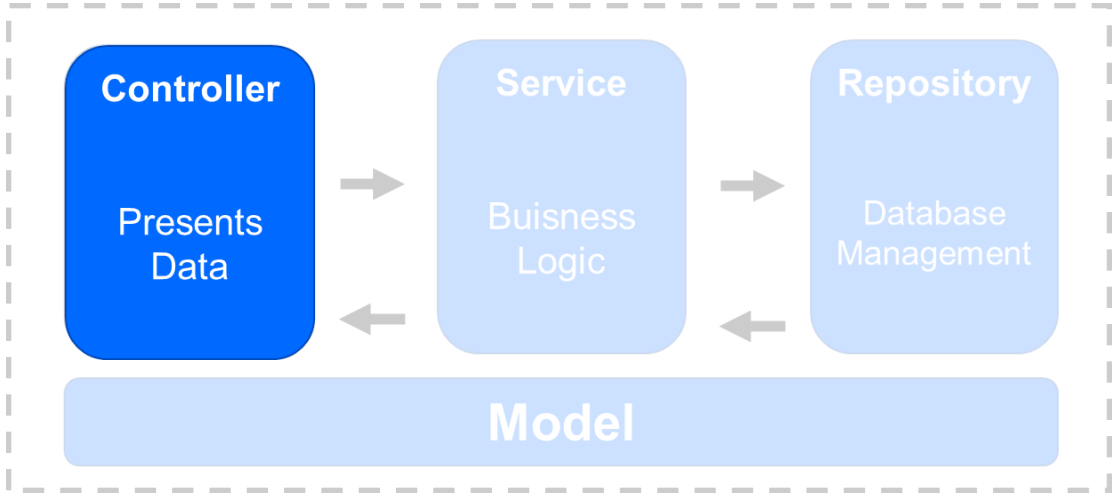


NOSERYOUNG

Controller Class

Repetition

- Implememnts REST endpoints
- Returns a response to the frontend
- Uses services to generate the response data

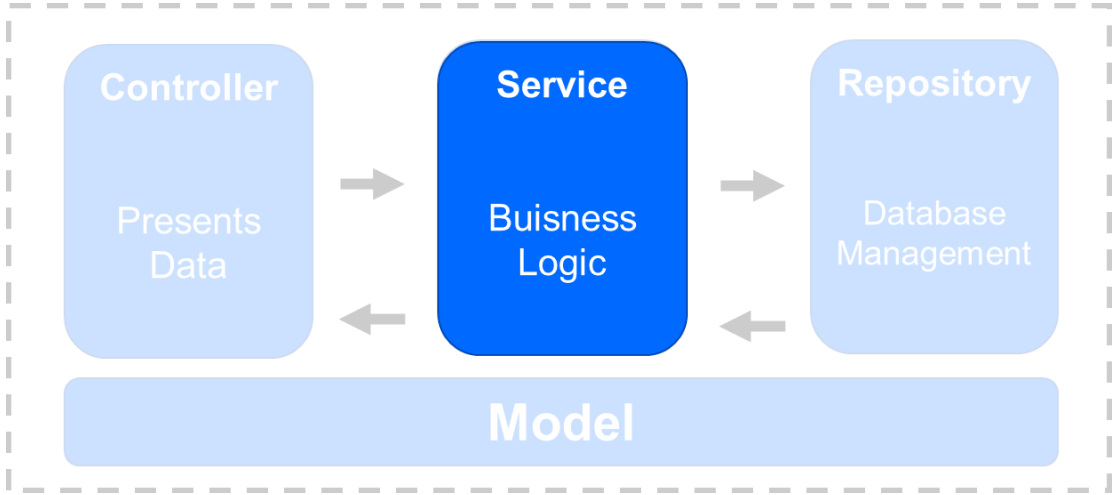


NOSERYOUNG

Service Class

Repetition

- Contains **buisness logic**
- Can interact with other services

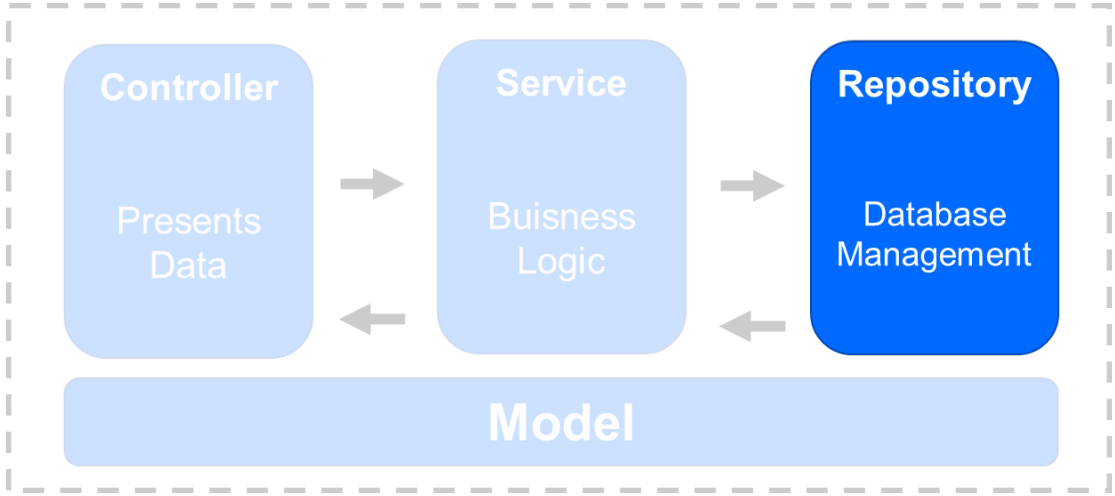


NOSERYOUNG

Repository Class

Repetition

- **Accesses Database** and provides access to other classes.
- Easy starting point: **Interface that inherits JpaRepository**
  - Standard CRUD operations are already implemented by Spring Data

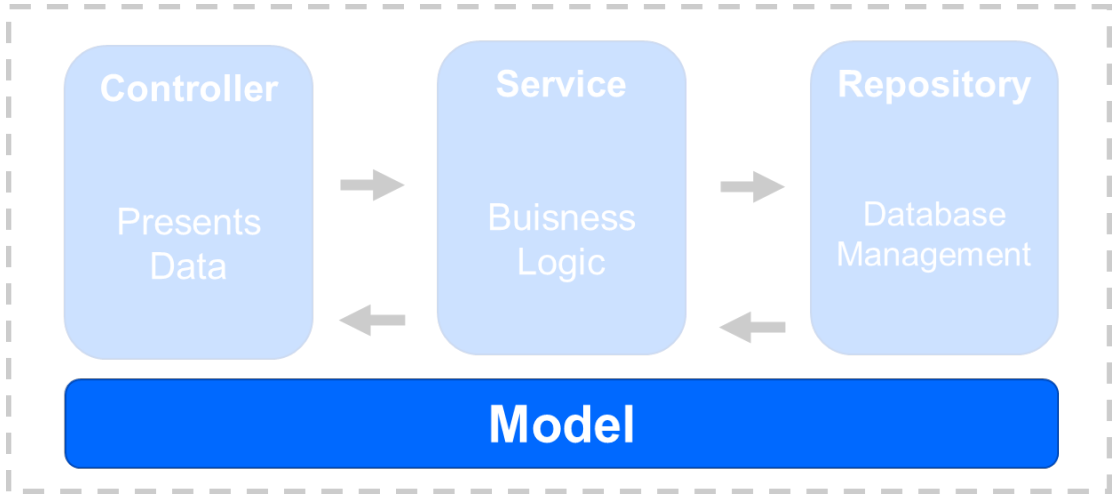


NOSERYOUNG

Model Class / Entitiy

Mapping Java Objects to Data tables

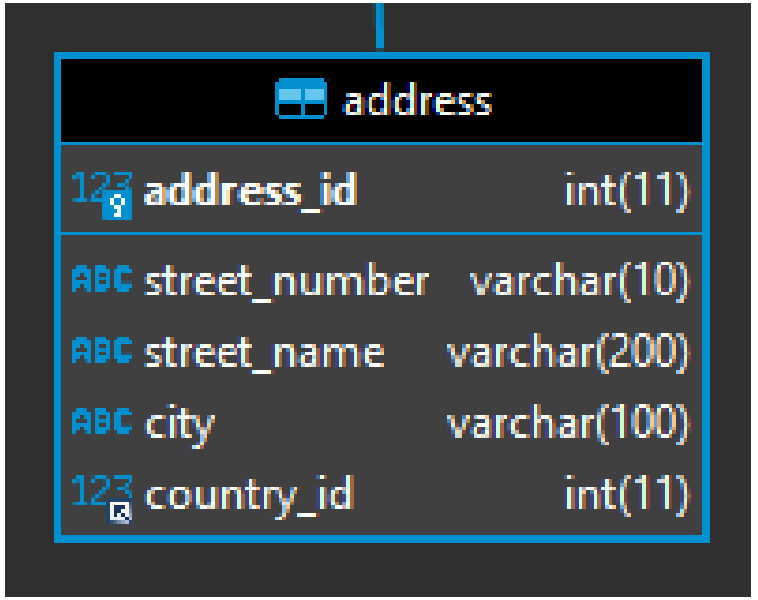
- **A Java-Class that is mapped to a table in a database**
- One model class for each (non-intermediate) table
- Class variables are columns of tables



NOSERYOUNG

DB Key Terms

- **Primary Key**: Unique identifier of a row in a table
- **Foreign Key**: Links tables together. References a primary key in another table
- **Cascade Delete**: Delete all References in other tables
- **Uni- vs. Bidirectional mapping**: References in one/ both direcitons



NOSERYOUNG

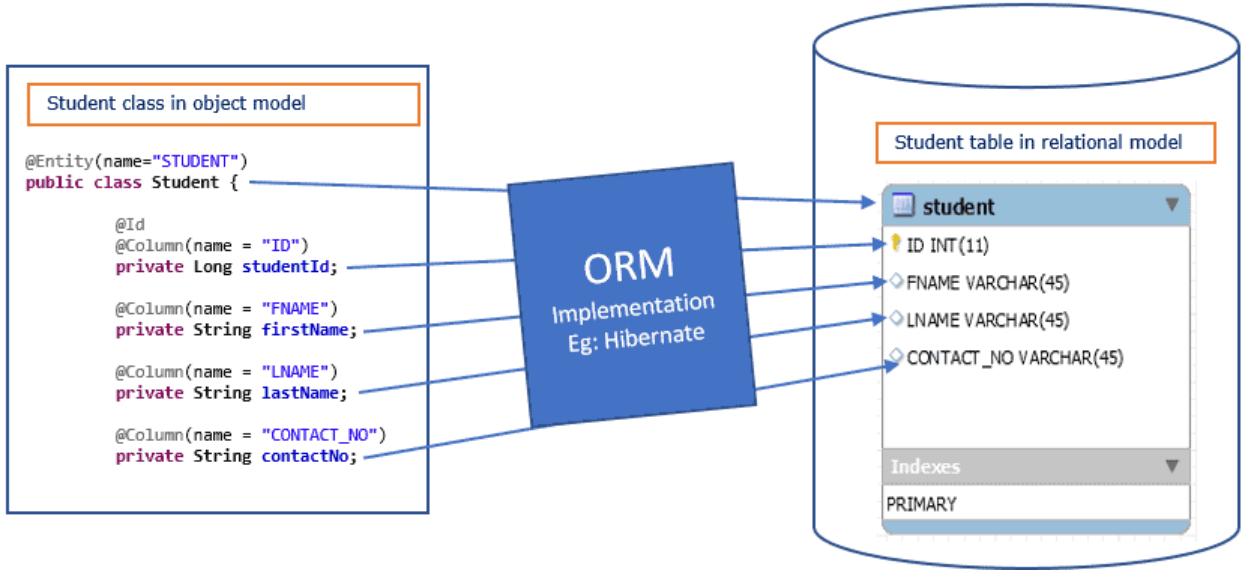
”

Object Relational Mappers

ORM

Object-Relational-Mapper

- Maps object-oriented code onto relational databases.
- Handles low level SQL code
  - Instead of working with SQL statements, developers can use object-oriented methods



NOSERYOUNG

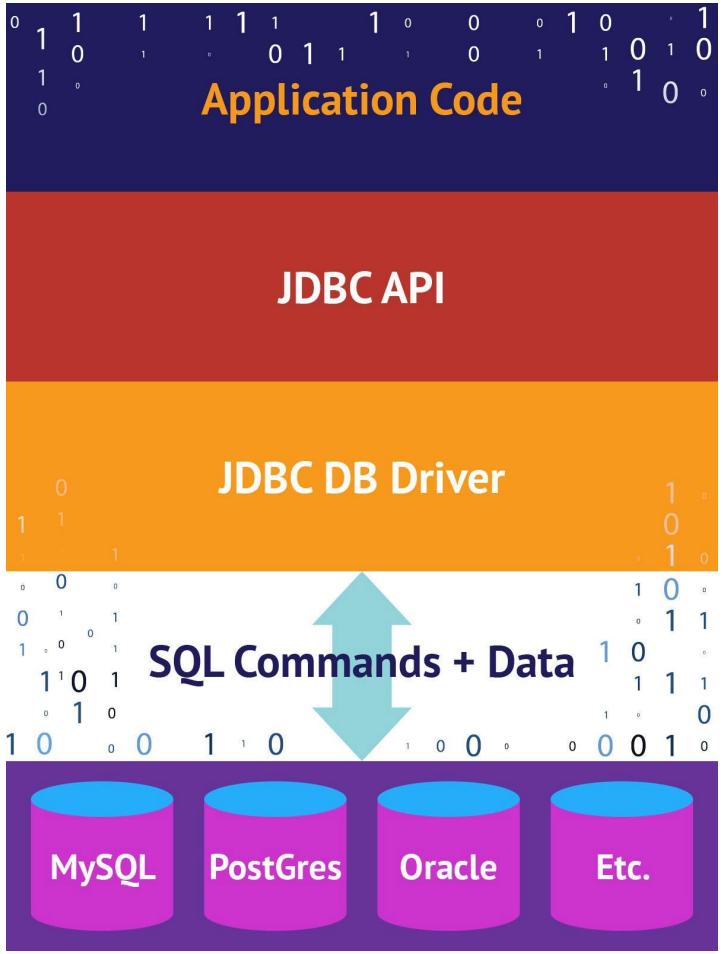
JDBC

Java Database Connectivity

- JDBC provides an API to connect a Java Applicaiton to a Database
- Developers write/program SQL-Queries

```
String QUERY = "select * from student_details";
```

- **Pros**: simple, controlled
- **Cons**: «hands on», not well-isolated, difficult to switch DB



NOSERYOUNG

JDBC

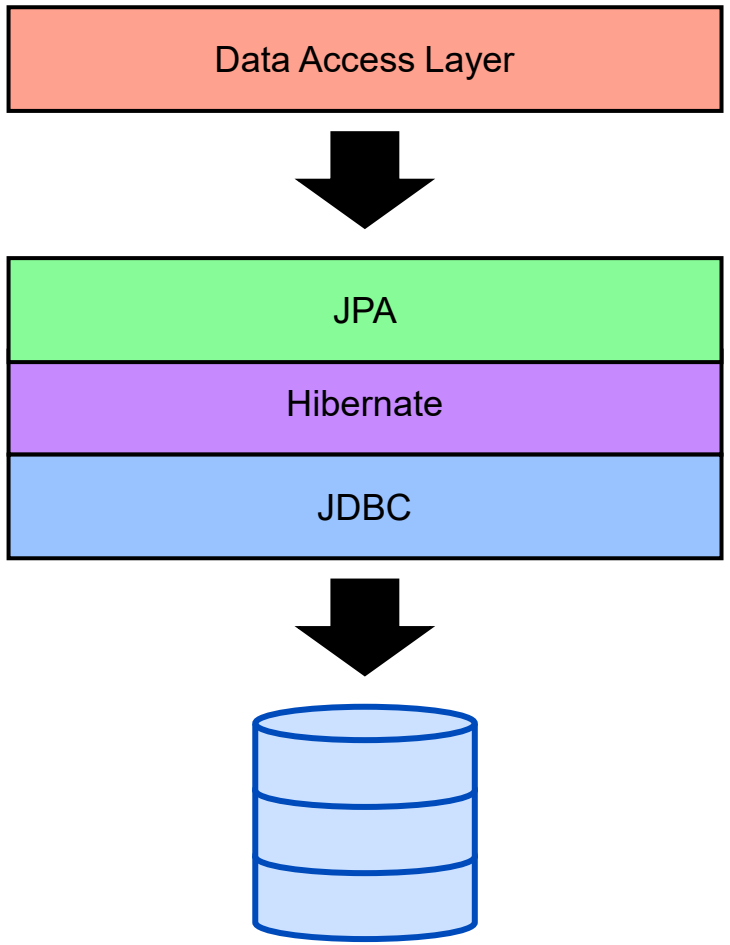
Example

```
import java.sql.Connection;

Connection connection = dataSource.getConnection(); // (1)

      try (connection) {
      connection.setAutoCommit(false); // (2)
      // execute some SQL statements...
      connection.commit(); // (3)

      } catch (SQLException e) {
      connection.rollback(); // (4)
      }
```

JPA

Java/Jakarta Persistence API

- An API for DB access and Object-Relational-Mapping (ORM).
- Separates SQL from your business logic. ➔ Higher levels of abstraction, less, coupling, easier to maintain.
- Portable across databases by default
- Higher overhead, more difficult to optimize.
- **JPA is implemented by different ORMs,** for example Hibernate



NOSERYOUNG

Using Spring Data

Model Class / Entitiy

Mapping Java Objects to Data tables

- **A Java-Class that is mapped to a table in a database**
- @Id: identifies variable as primary key
  - @GeneratedValue
- @Column: variable is mapped as a column o the table.
- @Transient: not saved in DB

```
@Entity
@Table (name="student")
public class Student {
   @Id
   @Column(name="id")
   private int id;

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

}
```

NOSERYOUNG

Relationships

Foreign Keys & Mappings

- @OneToOne (cascade={}, mappedBy=)
  - @JoinColumnName
- @ManyToOne
  - @JoinColumn
- @OneToMany (mappedBy=)
- @ManyToMany

Relationships in Spring Data

@JoinTable

- Optionally define parameters of join table
- It is specified on the **owning side** of an association.
- Typically used in many-to-many and unidirectional one-to-many associations

```
@JoinTable(
      name = "users_roles",
      joinColumns = @JoinColumn(
        name = "user_id",
         referencedColumnName = "id"),
      inverseJoinColumns =
        @JoinColumn(
            name = "role_id",
referencedColumnName = "id"))
private Set<Role> roles;
```

NOSERYOUNG

@Query

- This annotation allows us to write **custom SQL-queries**
- Can pass additional values, such as **sort- or pagination-information**

```
@Query(value = "SELECT u FROM User u")
List<User> findAllUsers()
```

@Query

- Can also generate SQL statements programmatically

```
@Modifying
@Query(
"update User u set u.status = :status where u.name = :name")
int updateUserSetStatusForName(
@Param("status") Integer status,
@Param("name") String name);
```

Eager Vs Lazy Loadings

- Lazy: Only load data when absolutely needed!
- `Fetch=FetchType.LAZY`
- **Attention!** Different Mappings have different default fetch-types:
  - …ToMany ➔ Lazy
  - …ToOne ➔ Eager

```
@OneToMany(mappedBy
="user",
fetch = FetchType.LAZY)
private Set<CourseUser>
courseUser;
```

@Transactional

- Wraps Java Statements into a **Transaction**.
- This allows us to manage the isolation, propagation and rollback behavior of JPA-methods

```
@Transactional(
      isolation =
      Isolation.READ_COMMITTED,
public void crudOperation() {
      readFromDatabase();
      writeToDatabase(); }
```

NOSERYOUNG

Rollback

@Transactional(rollbackFor, noRollbackFor)

- Defines the type of exceptions that cause the transaction to be rolled back
- Default: all Runtime Exceptions.
- `rollbackFor=Exception.class` rolls back for **any** Exception.

Pagination & Sorting

PageRequest & Sort

- An object of type **pagable** passed as argument to a **repository method** returns a page of the results.
- **Sort.by** defines the (set of) parameters to sort by.
- PageRequest and Sort **can be combined**

```
userRepository.findAll(
PageRequest.of(page,page_len))

userRepository.findAll(
Sort.by("username").descending()))
```

NOSERYOUNG

Documenting Transactions

Common Strategies

- Logging transactions in Spring: **Log4J & custom logging**

```
2012-08-22 18:50:00,031 TRACE - Getting transaction for
[com.MyClass.myMethod]
[my own log statements from method com.MyClass.myMethod]
2012-08-22 18:50:00,142 TRACE - Completing transaction for
[com.MyClass.myMethod]
```

- In PostgreSQL: **Write-Ahead Log (WAL)**
  - First write to log, then to DB

NOSERYOUNG

Good To know

AppStartupRunner

- To run commands after startup, e.g. to add some default user

```
@Component
@RequiredArgsConstructor

class AppStartupRunner
implements ApplicationRunner {
[…]
   @Override
   public void run(
ApplicationArguments args)
throws Exception {
}}
```

NOSERYOUNG

Good To know

Load Default / Test Data

- Load Data in DB on Startup by plaicng a sql file named «**data.sql**» in *src/main/resources*
- Script will be execued on startup
  - This option in *application.properties* might be necessary :

```
spring.jpa.defer-datasource-
initialization=true
```

Further Reading

- JPA Tutorial: https://spring.io/guides/gs/accessing-data-jpa/
- ORMs additional Info: https://javabydeveloper.com/orm-object-relational-mapping/
- Spring Data: https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

”

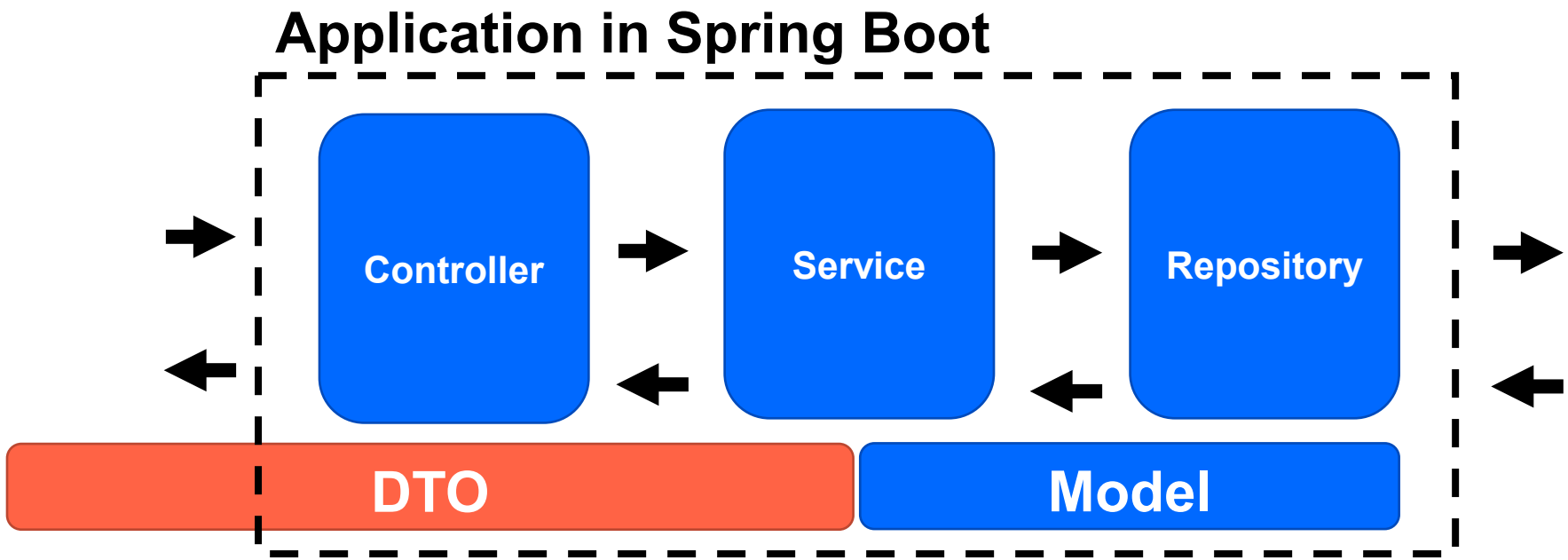
Advanced

NOSERYOUNG

Data Transfer Objects & Mappers

What?

- Object that is used to encapsulate data, and send it from one subsystem of an application to another.

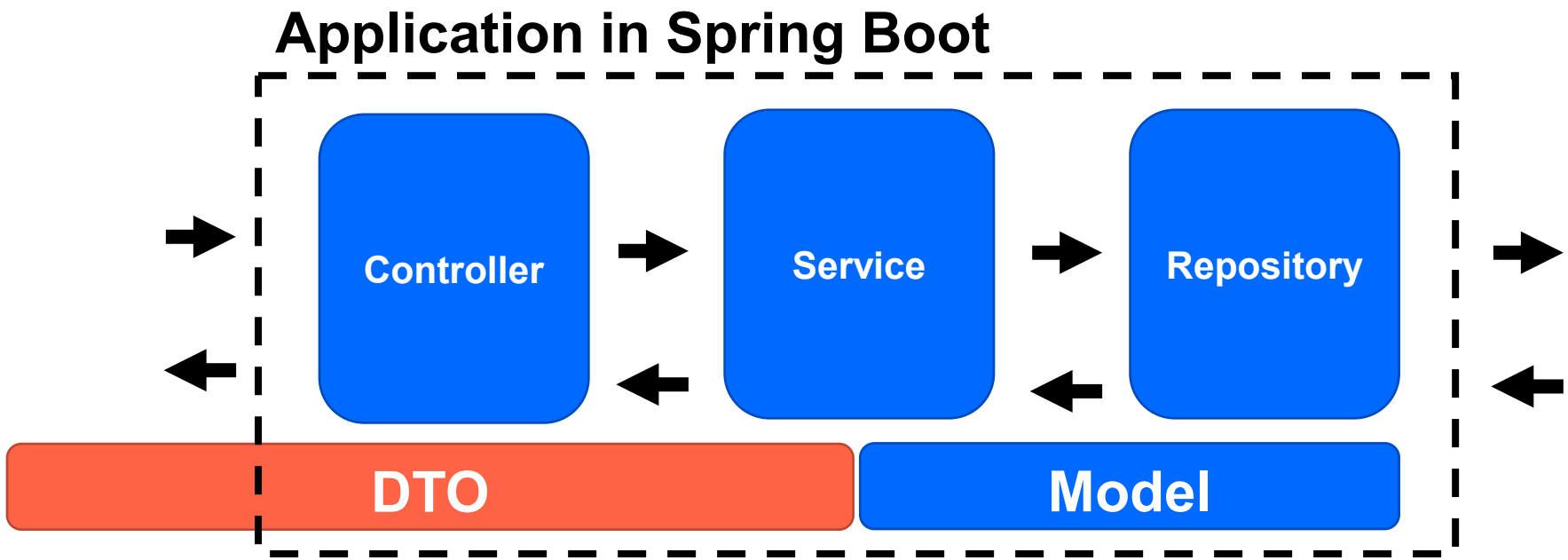


NOSERYOUNG

Data Transfer Objects & Mappers

How?

- **ModelMapper** for simpler single objects
- **MapStruct** for more complex applications, Lists etc.



NOSERYOUNG