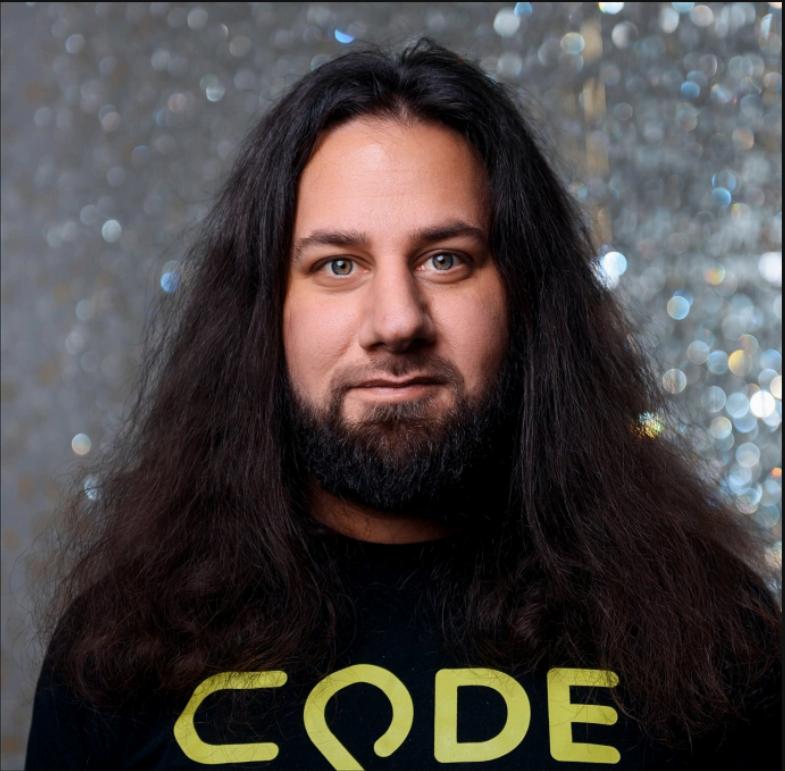


Advanced Kotlin Techniques for Spring Developers

Pasha Finkelshteyn 

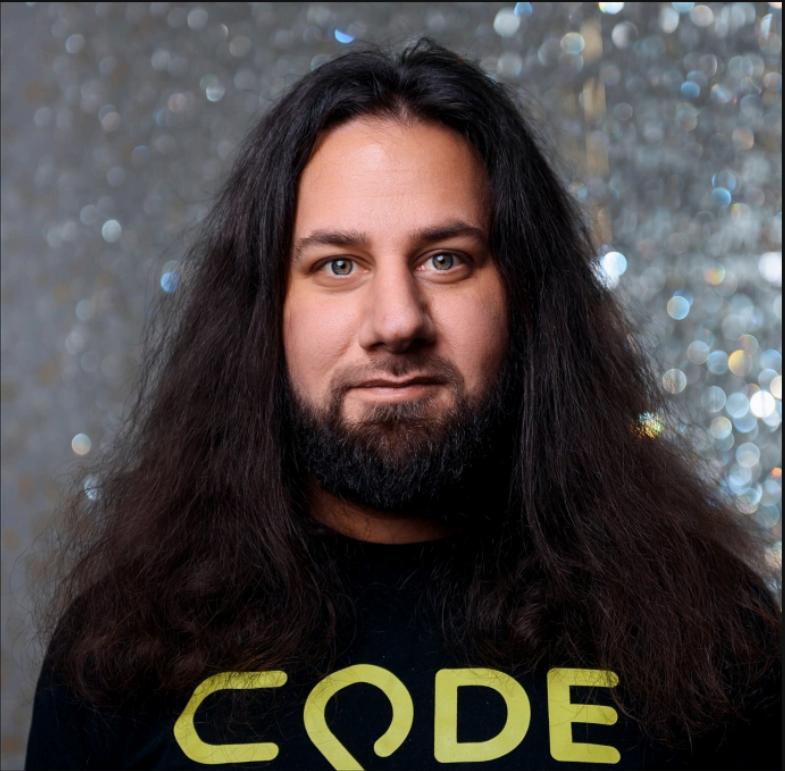
be^{ll}soft

whoami



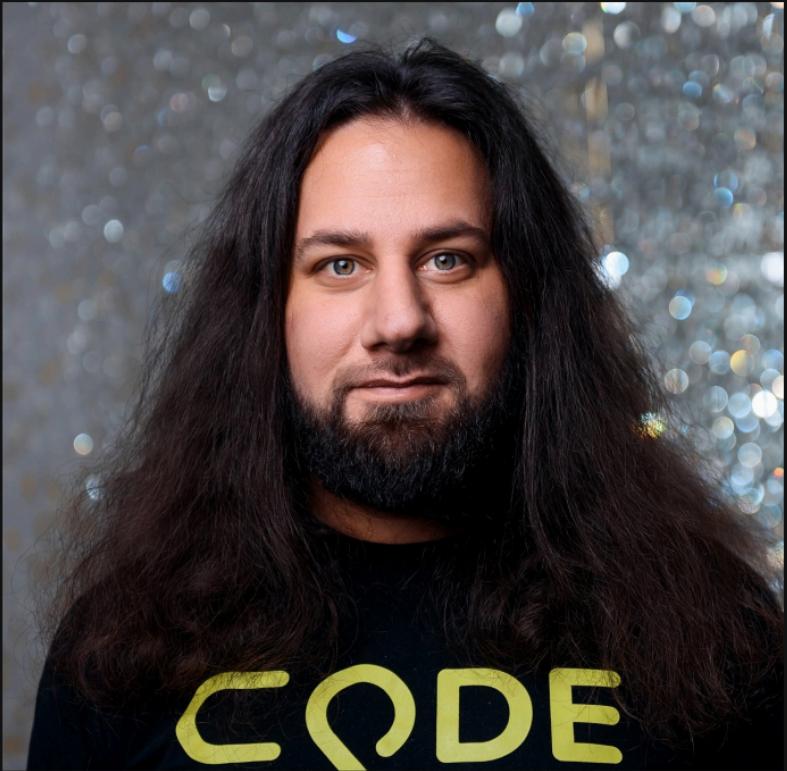
whoami

- Pasha Finkelshteyn



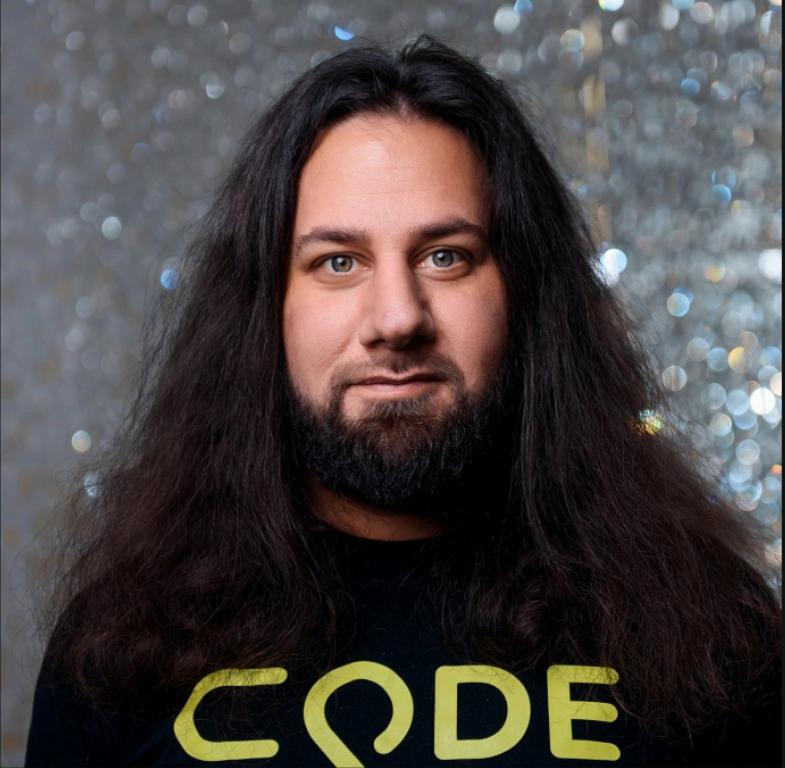
whoami

- Pasha Finkelshteyn
- Dev 🍅 at BellSoft



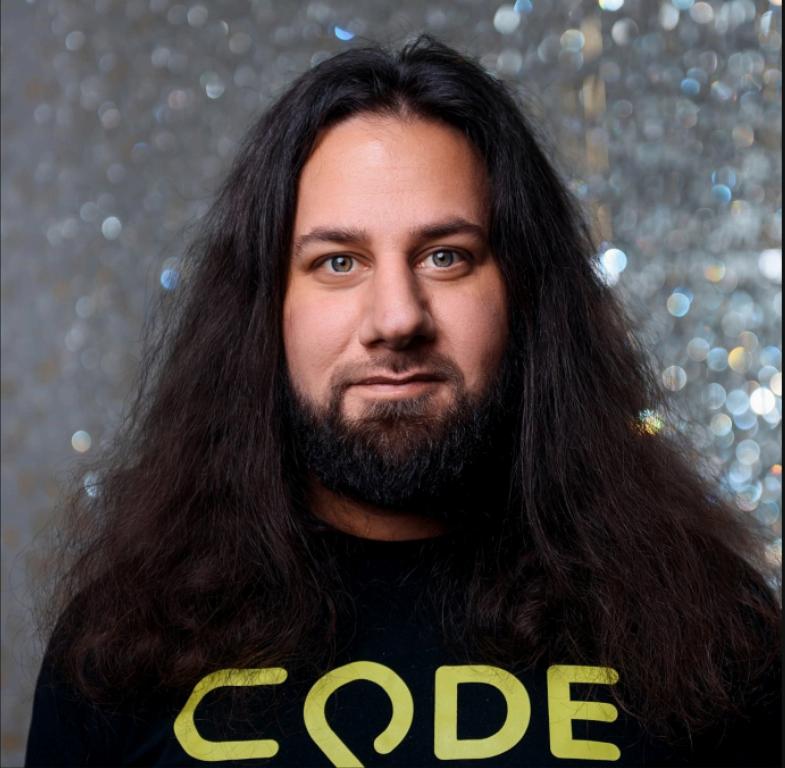
whoami

- Pasha Finkelshteyn
- Dev  at BellSoft
- ≈10 years in JVM. Mostly 
and 



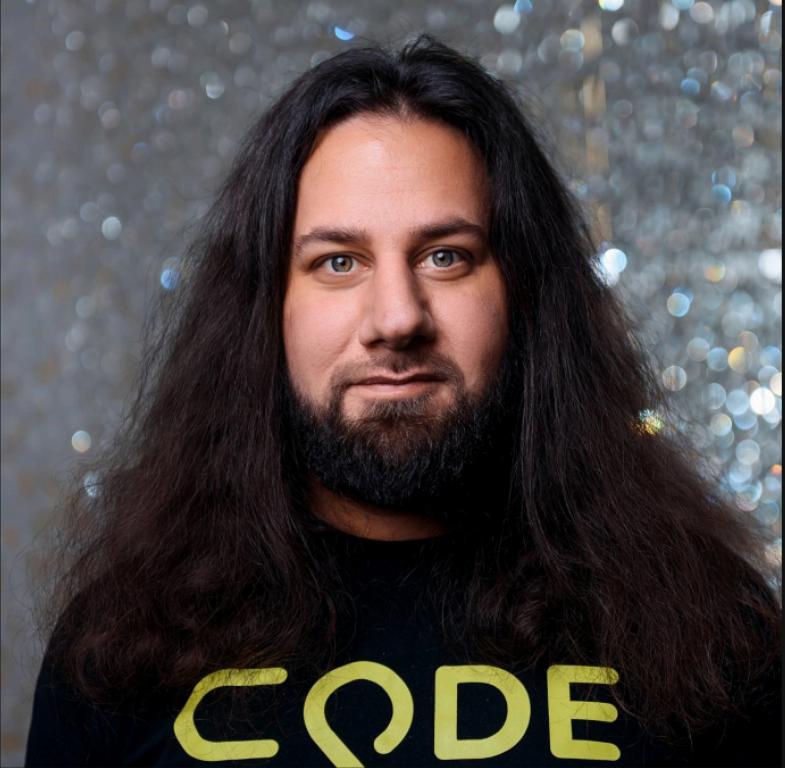
whoami

- Pasha Finkelshteyn
- Dev 🍅 at BellSoft
- ≈10 years in JVM. Mostly 🔥 and 💡
- And 🌱



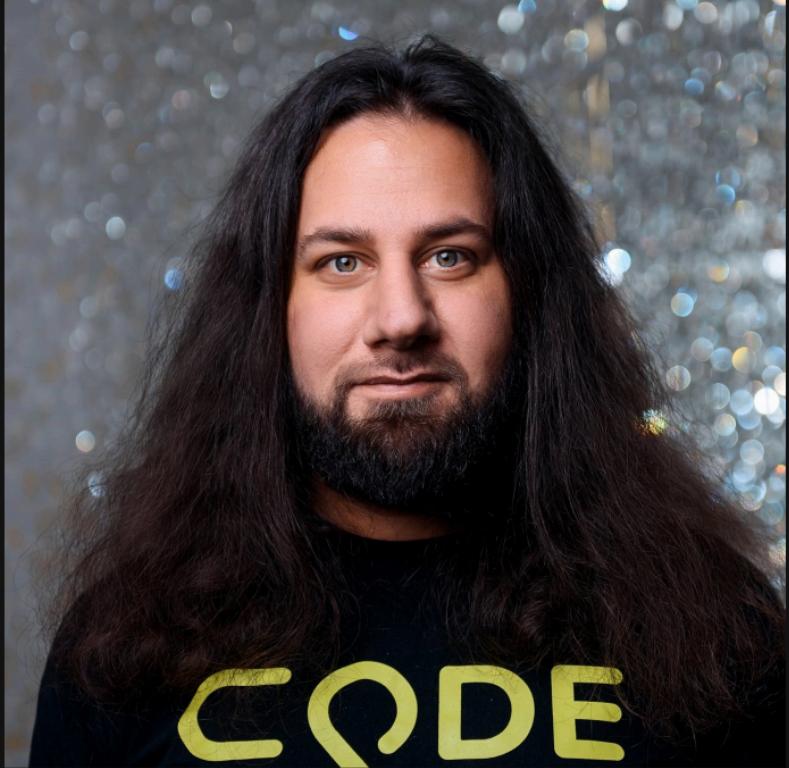
whoami

- Pasha Finkelshteyn
- Dev 🍅 at BellSoft
- ≈10 years in JVM. Mostly 🔥 and 💡
- And 💚
- 🐦 asm0di0



whoami

- Pasha Finkelshteyn
- Dev 🍄 at BellSoft
- ≈10 years in JVM. Mostly 🔥 and 💡
- And 🌱
- 🐦 asm0di0
- 🏡 @asm0dey@fosstodon.org



BellSoft

- Vendor of Liberica JDK
- Contributor to the OpenJDK
- Author of ARM32 support in JDK

Liberica is the JDK officially recommended by 



BellSoft

- Vendor of Liberica JDK
- Contributor to the OpenJDK
- Author of ARM32 support in JDK

Liberica is the JDK officially recommended by 

We know our stuff!



I hope to showcase
something you don't
know yet!

My application

My application

- Simple nano-service

My application

- Simple nano-service
- MVC

My application

- Simple nano-service
- MVC
- Validation

My application

- Simple nano-service
- MVC
- Validation
- JPA

My application

- Simple nano-service
- MVC
- Validation
- JPA
- JDBC

My application

- Simple nano-service
- MVC
- Validation
- JPA
- JDBC
- Tests

Where do I start?

<https://start.spring.io>

Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.1.1 (SNAPSHOT)
- 3.1.0
- 3.0.8 (SNAPSHOT)
- 3.0.7
- 2.7.13 (SNAPSHOT)
- 2.7.12

Minimum dependencies

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Validation I/O

Bean Validation with Hibernate validator.

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Security SECURITY

Highly customizable authentication and access-control framework for Spring applications.

PostgreSQL Driver SQL

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Testcontainers TESTING

Provide lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

2 files are generated

- build.gradle.kts
- SpringKotlinStartApplication.kt

What happens

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
    id("org.springframework.boot") version "3.3.0"
    id("io.spring.dependency-management") version "1.1.5"
    kotlin("jvm") version "1.9.24"
    kotlin("plugin.spring") version "1.9.24"
    kotlin("plugin.jpa") version "1.9.24"
}

group = "com.github.asm0dey"
version = "0.0.1-SNAPSHOT"

java {
    sourceCompatibility = JavaVersion.VERSION_21
}

repositories {
    mavenCentral()
```

The main class

Main class

```
1 import org.springframework.boot.autoconfigure.SpringBootApplication
2 import org.springframework.boot.runApplication
3
4 @SpringBootApplication
5 class SampleApplication
6
7 fun main(args: Array<String>) {
8     runApplication<SampleApplication>(*args)
9 }
```

Main class

```
1 import org.springframework.boot.autoconfigure.SpringBootApplication
2 import org.springframework.boot.runApplication
3
4 @SpringBootApplication
5 class SampleApplication
6
7 fun main(args: Array<String>) {
8     runApplication<SampleApplication>(*args)
9 }
```

Main class

```
1 import org.springframework.boot.autoconfigure.SpringBootApplication
2 import org.springframework.boot.runApplication
3
4 @SpringBootApplication
5 class SampleApplication
6
7 fun main(args: Array<String>) {
8     runApplication<SampleApplication>(*args)
9 }
```

runApplication

```
1  inline fun <reified T : Any> runApplication(vararg args: String) =  
2      SpringApplication.run(T::class.java, *args)
```

runApplication

```
1 inline fun <reified T : Any> runApplication(vararg args: String) =  
2     SpringApplication.run(T::class.java, *args)
```

runApplication

```
1 inline fun <reified T : Any> runApplication(vararg args: String) =  
2     SpringApplication.run(T::class.java, *args)
```

runApplication

```
1 inline fun <reified T : Any> runApplication(vararg args: String) =  
2     SpringApplication.run(T::class.java, *args)
```

The first goodie of Spring for Kotlin

Let's start implementing

Chapter 1. MVC + Validation

First controller

```
1  @RestController
2  @RequestMapping("/person")
3  class PersonController {
4      @PostMapping
5      fun createPerson(@RequestBody @Valid person: Person) {}
6  }
```

First controller

```
1  @RestController
2  @RequestMapping("/person")
3  class PersonController {
4      @PostMapping
5      fun createPerson(@RequestBody @Valid person: Person) {}
6  }
```

First controller

```
1  @RestController
2  @RequestMapping("/person")
3  class PersonController {
4      @PostMapping
5      fun createPerson(@RequestBody @Valid person: Person) {}
6  }
```

First controller

```
1  @RestController
2  @RequestMapping("/person")
3  class PersonController {
4      @PostMapping
5      fun createPerson(@RequestBody @Valid person: Person) {}
6  }
```

First controller

```
1  @RestController
2  @RequestMapping("/person")
3  class PersonController {
4      @PostMapping
5      fun createPerson(@RequestBody @Valid person: Person) {}
6  }
```

Person.kt :

```
data class Person(
    val name: String,
    val age: Int
)
```

Make an empty POST ...

```
1 POST localhost:8080/person  
2 Content-Type: application/json
```

Make an empty POST ...

```
1 POST localhost:8080/person  
2 Content-Type: application/json
```

Make an empty POST ...

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
3 {
4     "timestamp": 1674735741056,
5     "status": 400,
6     "error": "Bad Request",
7     "path": "/person"
8 }
```

Make an empty POST ...

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
3 {
4     "timestamp": 1674735741056,
5     "status": 400,
6     "error": "Bad Request",
7     "path": "/person"
8 }
```

Make an empty POST ...

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
3 {
4     "timestamp": 1674735741056,
5     "status": 400,
6     "error": "Bad Request",
7     "path": "/person"
8 }
```

Since `Person` is non-nullable — it's validated without `@NotNull` annotation

Why? How?

build.greadle.kts :

```
1 tasks.withType<KotlinCompile> {
2     kotlinOptions {
3         freeCompilerArgs = listOf("-Xjsr305=strict")
4         jvmTarget = "21"
5     }
6 }
```

Why? How?

build.greadle.kts :

```
1 tasks.withType<KotlinCompile> {
2     kotlinOptions {
3         freeCompilerArgs = listOf("-Xjsr305=strict")
4         jvmTarget = "21"
5     }
6 }
```

Why? How?

build.greadle.kts :

```
1 tasks.withType<KotlinCompile> {
2     kotlinOptions {
3         freeCompilerArgs = listOf("-Xjsr305=strict")
4         jvmTarget = "21"
5     }
6 }
```

Why? How?

build.gradle.kts :

```
1 tasks.withType<KotlinCompile> {  
2     kotlinOptions {  
3         freeCompilerArgs = listOf("-Xjsr305=strict")  
4         jvmTarget = "21"  
5     }  
6 }
```

JSR 305: Annotations for Software Defect Detection:

Nullness annotations (e.g., `@NonNull` and `@CheckForNull`)

Internationalization annotations, such as `@NonNls` or `@Nls`

Non-empty POST with empty properties

```
1  POST localhost:8080/person
2  Content-Type: application/json
3
4  {"name": null, "age": null}
```

Non-empty POST with empty properties

```
1  POST localhost:8080/person
2  Content-Type: application/json
3
4  {"name": null, "age": null}
```

On client

Non-empty POST with empty properties

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": null, "age": null}
```

On client

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
```

Non-empty POST with empty properties

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": null, "age": null}
```

On client

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
```

On server

Non-empty POST with empty properties

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": null, "age": null}
```

On client

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
```

On server

```
1 ...Instantiation of [simple type, class com.github.asm0dey.sample.Person]
2     value failed for JSON property name due to missing
```

Non-empty POST with empty properties

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": null, "age": null}
```

On client

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
```

On server

```
1 ...Instantiation of [simple type, class com.github.asm0dey.sample.Person]
2     value failed for JSON property name due to missing
```

POST with non-empty name

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": "Pasha", "age": null}
```

POST with non-empty name

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": "Pasha", "age": null}
```

POST with non-empty name

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": "Pasha", "age": null}
```

```
1 HTTP/1.1 200
2 Content-Length: 0
```

POST with non-empty name

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": "Pasha", "age": null}
```

```
1 HTTP/1.1 200
2 Content-Length: 0
```

POST with non-empty name

```
1 POST localhost:8080/person
2 Content-Type: application/json
3
4 {"name": "Pasha", "age": null}
```

```
1 HTTP/1.1 200
2 Content-Length: 0
```



Rechecking

```
1  data class Person(  
2      val name: String,  
3      val age: Double  
4  )
```

Rechecking

```
1  data class Person(  
2      val name: String,  
3      val age: Double  
4  )
```

Rechecking

```
1  data class Person(  
2      val name: String,  
3      val age: Double  
4  )
```



In JVM primitive types
have default values

These types will be JVM primitives:

- Double
- Int
- Float
- Char
- Short
- Byte
- Boolean

Updating class

```
1  data class Person(  
2      val name: String,  
3      @field:NotNull val age: Double?  
4  )
```

Updating class

```
1  data class Person(  
2      val name: String,  
3      @field:NotNull val age: Double?  
4  )
```

Updating class

```
1 data class Person(  
2     val name: String,  
3     @field:NotNull val age: Double?  
4 )
```

```
1 POST localhost:8080/person  
2 Content-Type: application/json  
3  
4 {"name": "Pasha", "age": null}
```

Updating class

```
1 data class Person(  
2     val name: String,  
3     @field:NotNull val age: Double?  
4 )
```

```
1 POST localhost:8080/person  
2 Content-Type: application/json  
3  
4 {"name": "Pasha", "age": null}
```

Updating class

```
1 data class Person(  
2     val name: String,  
3     @field:NotNull val age: Double?  
4 )
```

```
1 POST localhost:8080/person  
2 Content-Type: application/json  
3  
4 {"name": "Pasha", "age": null}
```

```
1 HTTP/1.1 400 Bad Request  
2 ...  
3 { "timestamp": 1674760360096, "status": 400, "error": "Bad Request", "path": "/person" }
```

Updating class

```
1 data class Person(  
2     val name: String,  
3     @field:NotNull val age: Double?  
4 )
```

```
1 POST localhost:8080/person  
2 Content-Type: application/json  
3  
4 {"name": "Pasha", "age": null}
```

```
1 HTTP/1.1 400 Bad Request  
2 ...  
3 { "timestamp": 1674760360096, "status": 400, "error": "Bad Request", "path": "/person" }
```

```
1 Field error in object 'person' on field 'age': rejected value [null]
```

Also...

```
1  spring:
2    jackson:
3      deserialization:
4        FAIL_ON_NULL_FOR_PRIMITIVES: true
```

Quick summary

- `-Xjsr305=strict` will make the validation easier
- For JVM primitive types we have to put `@field:NotNull` and mark them nullable
- Sometimes can work it around with jackson settings

JPA

Chapter 2

Nanoentity

```
1  @Entity
2  data class Person(
3      @Id
4      @GeneratedValue(strategy = IDENTITY)
5      var id: Int? = null,
6      @Column(nullable = false)
7      val name: String,
8      @Column(nullable = false)
9      val age: Int,
10     )
```

Nanoentity

```
1  @Entity
2  data class Person(
3      @Id
4      @GeneratedValue(strategy = IDENTITY)
5      var id: Int? = null,
6      @Column(nullable = false)
7      val name: String,
8      @Column(nullable = false)
9      val age: Int,
10     )
```

- `data class`

Nanoentity

```
1  @Entity
2  data class Person(
3      @Id
4      @GeneratedValue(strategy = IDENTITY)
5      var id: Int? = null,
6      @Column(nullable = false)
7      val name: String,
8      @Column(nullable = false)
9      val age: Int,
10     )
```

- `data class`
- `val name` and `val age`

Nanoentity

```
1  @Entity
2  data class Person(
3      @Id
4      @GeneratedValue(strategy = IDENTITY)
5      var id: Int? = null,
6      @Column(nullable = false)
7      val name: String,
8      @Column(nullable = false)
9      val age: Int,
10     )
```

- `data class`
- `val name` and `val age`

Improving

data classes have copy, equals, hashCode, copy, and componentX defined

```
@Entity
data class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    val name: String,
    @Column(nullable = false)
    val age: Int,
)
```

Improving

data classes have copy, equals, hashCode, copy, and componentX defined

```
@Entity
data class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    val name: String,
    @Column(nullable = false)
    val age: Int,
)
```

Improving

```
@Entity
class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    val name: String,
    @Column(nullable = false)
    val age: Int,
)
```

Improving

```
@Entity
class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    val name: String,
    @Column(nullable = false)
    val age: Int,
)
```

JPA won't be able to write to `val`

Improving

```
@Entity
class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    var name: String,
    @Column(nullable = false)
    var age: Int,
)
```

JPA won't be able to write to `val`

Improving

```
@Entity
class Person(
    @Id
    @GeneratedValue(strategy = IDENTITY)
    var id: Int? = null,
    @Column(nullable = false)
    var name: String,
    @Column(nullable = false)
    var age: Int,
)
```

JPA won't be able to write to `val`

But there is no no-arg constructor!

How to make it work?

Magic:

```
1  kotlin("plugin.jpa") version "1.8.0"
```

But there is no no-arg constructor!

How to make it work?

Magic:

```
1  kotlin("plugin.jpa") version "1.8.0"
```

- Puts annotations on the fields
- Adds a default constructor in bytecode*!

But there is no no-arg constructor!

How to make it work?

Magic:

```
1  kotlin("plugin.jpa") version "1.8.0"
```

- Puts annotations on the fields
- Adds a default constructor in bytecode*!

* In Kotlin the default constructor would not be possible, but in Java it is

Current result

```
1  @Entity
2  class Person(
3      @Id
4      @GeneratedValue(strategy = IDENTITY)
5      var id: Int? = null,
6      @Column(nullable = false)
7      var name: String,
8      @Column(nullable = false)
9      var age: Int,
10     )
```

Is this enough?

Not quite.

At the very least we have to redefine `equals` and `hashCode`.

For example...

```
1  @Entity
2  class Person(
3      // properties
4  ) {
5      // equals...
6      override fun hashCode(): Int {
7          return id ?: 0
8      }
9  }
```

Is this enough?

Not quite.

At the very least we have to redefine `equals` and `hashCode`.

For example...

```
1  @Entity
2  class Person(
3      // properties
4  ) {
5      // equals...
6      override fun hashCode(): Int {
7          return id ?: 0
8      }
9  }
```

Is this enough?

Not quite.

At the very least we have to redefine `equals` and `hashCode`.

For example...

```
1  @Entity
2  class Person(
3      // properties
4  ) {
5      // equals...
6      override fun hashCode(): Int {
7          return id ?: 0
8      }
9  }
```

Is this enough?

Not quite.

At the very least we have to redefine `equals` and `hashCode`.

For example...

```
1  @Entity
2  class Person(
3      // properties
4  ) {
5      // equals...
6      override fun hashCode(): Int {
7          return id ?: 0
8      }
9  }
```

JDBC

Chapter 3

Obtain user by id

Let's imagine we need to call the following:

```
1  SELECT *
2  FROM users
3  WHERE id = ?
```

Or, for example...

Or, for example...

In Java



Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java



Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int id) {
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);
}

private static class UserRowMapper implements RowMapper<Person> {
    @Override
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {
        int id = resultSet.getInt("id");
        String name = resultSet.getString("name");
        Double age = resultSet.getDouble("age");
        return new Person(id, name, age);
    }
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java



Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java



Let's inline mapper

```
public List<Person> findById(int id) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", new UserRowMapper(), id);  
}  
  
private static class UserRowMapper implements RowMapper<Person> {  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }  
}
```

In Java



Let's inline mapper

```
public List<Person> findById(int userId) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", (resultSet, i) -> {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }, userId);  
}
```

In Java

☕ Let's inline mapper

```
public List<Person> findById(int userId) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", (resultSet, i) -> {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }, userId);  
}
```

In Java



```
public List<Person> findById(int userId) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", (resultSet, i) -> {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }, userId);  
}
```

In Java



```
public List<Person> findById(int userId) {  
    return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", (resultSet, i) -> {  
        int id = resultSet.getInt("id");  
        String name = resultSet.getString("name");  
        Double age = resultSet.getDouble("age");  
        return new Person(id, name, age);  
    }, userId);  
}
```

I don't like it

- Too many mappers
- Parameters are too far from query



Why should it be so?

Why should it be so?

Let's Look at the signature

Why should it be so?

Let's Look at the signature

```
1  public <T> List<T> query(String sql, RowMapper<T> rowMapper, @Nullable Object... args)
```

Why should it be so?

Let's Look at the signature

```
1  public <T> List<T> query(String sql, RowMapper<T> rowMapper, @Nullable Object... args)
```

Because in ☕ vararg can be only the last... 😢

JdbcTemplate in Kotlin



```
1 return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", userId) { rs, _ ->
2     val id = rs.getInt("id")
3     val name = rs.getString("name")
4     val age = rs.getDouble("age")
5     Person(id, name, age)
6 }
```

JdbcTemplate in Kotlin



```
1 return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", userId) { rs, _ ->
2     val id = rs.getInt("id")
3     val name = rs.getString("name")
4     val age = rs.getDouble("age")
5     Person(id, name, age)
6 }
```

JdbcTemplate in Kotlin



```
1 return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", userId) { rs, _ ->
2     val id = rs.getInt("id")
3     val name = rs.getString("name")
4     val age = rs.getDouble("age")
5     Person(id, name, age)
6 }
```

JdbcTemplate in Kotlin



```
1 return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", userId) { rs, _ ->
2     val id = rs.getInt("id")
3     val name = rs.getString("name")
4     val age = rs.getDouble("age")
5     Person(id, name, age)
6 }
```

- `vararg` doesn't have to be in the last position
- unused parameter of a lambda can be named `_`

Extension functions

```
1 fun <T> JdbcOperations.query(  
2     sql: String,  
3     vararg args: Any,  
4     function: (ResultSet, Int) -> T  
5 ): List<T>
```

Extension functions

```
1 fun <T> JdbcOperations.query(  
2     sql: String,  
3     vararg args: Any,  
4     function: (ResultSet, Int) -> T  
5 ): List<T>
```

Extension functions

```
1 fun <T> JdbcOperations.query(  
2     sql: String,  
3     vararg args: Any,  
4     function: (ResultSet, Int) -> T  
5 ): List<T>
```

Extension functions

```
1  fun <T> JdbcOperations.query(  
2      sql: String,  
3      vararg args: Any,  
4      function: (ResultSet, Int) -> T  
5  ): List<T>
```

Extension functions

```
1 fun <T> JdbcOperations.query(  
2     sql: String,  
3     vararg args: Any,  
4     function: (ResultSet, Int) -> T  
5 ): List<T>
```

Extension functions

```
1 fun <T> JdbcOperations.query(  
2     sql: String,  
3     vararg args: Any,  
4     function: (ResultSet, Int) -> T  
5 ): List<T>
```

Which allows

```
1 return jdbcTemplate.query("SELECT * FROM users WHERE id = ?", userId)  
2 { rs, _ ->  
3     // TODO: ResultSet → Person  
4 }
```

You can do the same for your own code!

From 

```
1  public List<String> transformStrings(Function<String, String> mapper, String... args){}
```

You can do the same for your own code!

From 

```
1 public List<String> transformStrings(Function<String, String> mapper, String... args){}
```

To 

```
1 fun transformStrings(vararg args: String, mapper: (String) -> String): List<String>{}
```

More on extensions for Spring

- [spring-beans](#)
- [spring-context](#)
- [spring-core](#)
- [spring-jdbc](#)
- [spring-messaging](#)
- [spring-r2dbc](#)
- [spring-test](#)
- [spring-tx](#)
- [spring-web](#)
- [spring-webflux](#)
- [spring-webmvc](#)

Reactive persistence

Chapter 4

Who knows what is reactive?



Who knows what is reactive?

- non-blocking



Who knows what is reactive?

- non-blocking
- asynchronous



Who knows what is reactive?

- non-blocking
- asynchronous
- handles back-pressure



Who knows what is reactive?

- non-blocking
- asynchronous
- handles back-pressure

The result is a monad...



Who knows what is reactive?

- non-blocking
- asynchronous
- handles back-pressure

The result is either:

- Mono : produces 0 to 1 items when it's ready
- Flux : Produces 0 to ∞ items



In Kotlin

Coroutines operate on `suspend` functions

`suspend` can be called from:

- coroutine context
- another `suspend fun`

returns usual types - Lists, Strings, etc

```
1  suspend fun randomNum(): Int {  
2      val x = service1.randomNum() // suspend  
3      val y = service2.randomNum() // suspend  
4      return x + y  
5  }
```

In Kotlin

Coroutines operate on `suspend` functions

`suspend` can be called from:

- coroutine context
- another `suspend fun`

returns usual types - Lists, Strings, etc

```
1  suspend fun randomNum(): Int {  
2      val x = service1.randomNum() // suspend  
3      val y = service2.randomNum() // suspend  
4      return x + y  
5  }
```

In Kotlin

Coroutines operate on `suspend` functions

`suspend` can be called from:

- coroutine context
- another `suspend fun`

returns usual types - Lists, Strings, etc

```
1  suspend fun randomNum(): Int {  
2      val x = service1.randomNum() // suspend  
3      val y = service2.randomNum() // suspend  
4      return x + y  
5  }
```

In Kotlin

Coroutines operate on `suspend` functions

`suspend` can be called from:

- coroutine context
- another `suspend fun`

returns usual types - Lists, Strings, etc

```
1  suspend fun randomNum(): Int {  
2      val x = service1.randomNum() // suspend  
3      val y = service2.randomNum() // suspend  
4      return x + y  
5  }
```

SUSPEND



imgflip.com

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long on not returned")
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long on not returned")
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long or not returned")
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long or not returned")
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long on not returned")
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long or not returned")
    // inside Spring ↓
    suspend fun <T> RowsFetchSpec<T>.awaitSingle(): T {
        return first().awaitSingleOrNull() ?: throw EmptyResultDataAccessException(1)
    }
}
```

Spring R2DBC

```
@Repository
class Repo(connectionFactory: ConnectionFactory) {
    val client = DatabaseClient.create(connectionFactory)
    suspend fun createUserAndReturnId() =
        client
            .sql("INSERT INTO users (name, email, age) VALUES ('Pasha', :email, NULL) RETURNING id")
            .bind("email", RandomStringUtils.randomAlphabetic(20))
            .fetch()
            .awaitSingle()["id"] as? Long ?: error("not long or not returned")
}
```

Transactions

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val curId = repo.createUserAndReturnId()
4      repo.createUserWithConflictingId(curId) // will throw
5      return curId
6  }
```

Transactions

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val curId = repo.createUserAndReturnId()
4      repo.createUserWithConflictingId(curId) // will throw
5      return curId
6  }
```

Transactions

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val curId = repo.createUserAndReturnId()
4      repo.createUserWithConflictingId(curId) // will throw
5      return curId
6  }
```

Transactions

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val curId = repo.createUserAndReturnId()
4      repo.createUserWithConflictingId(curId) // will throw
5      return curId
6  }
```

Transactions

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val curId = repo.createUserAndReturnId()
4      repo.createUserWithConflictingId(curId) // will throw
5      return curId
6  }
```

Nothing changes from the client standpoint!

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
}  
// Inside Spring  
interface CoroutineCrudRepository<T, ID> : Repository<T, ID> {  
    // ...  
    suspend fun <S : T> save(entity: S): T  
    // Flow is an async unbounded collection  
    fun findAll(): Flow<T>  
    // ...  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
}  
// Inside Spring  
interface CoroutineCrudRepository<T, ID> : Repository<T, ID> {  
    // ...  
    suspend fun <S : T> save(entity: S): T  
    // Flow is an async unbounded collection  
    fun findAll(): Flow<T>  
    // ...  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
}  
// Inside Spring  
interface CoroutineCrudRepository<T, ID> : Repository<T, ID> {  
    // ...  
    suspend fun <S : T> save(entity: S): T  
    // Flow is an async unbounded collection  
    fun findAll(): Flow<T>  
    // ...  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
    suspend fun findOne(id: String): User  
    fun findByFirstname(firstname: String): Flow<User>  
    suspend fun findAllByFirstname(id: String): List<User>  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
    suspend fun findOne(id: String): User  
    fun findByFirstname(firstname: String): Flow<User>  
    suspend fun findAllByFirstname(id: String): List<User>  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
    suspend fun findOne(id: String): User  
    fun findByFirstname(firstname: String): Flow<User>  
    suspend fun findAllByFirstname(id: String): List<User>  
}
```

Repositories

```
interface User : CoroutineCrudRepository<User, Long> {  
    suspend fun findOne(id: String): User  
    fun findByFirstname(firstname: String): Flow<User>  
    suspend fun findAllByFirstname(id: String): List<User>  
}
```

Function is either suspend or returns `Flow`

Transactions with repositories?

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val u = User(null, "me@asm0dey.site", 37)
4      val savedId = repo.save(u)
5      repo.save(u.copy(id = savedId)) // will throw
6      return curId
7  }
```

Transactions with repositories?

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val u = User(null, "me@asm0dey.site", 37)
4      val savedId = repo.save(u)
5      repo.save(u.copy(id = savedId)) // will throw
6      return curId
7  }
```

Transactions with repositories?

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val u = User(null, "me@asm0dey.site", 37)
4      val savedId = repo.save(u)
5      repo.save(u.copy(id = savedId)) // will throw
6      return curId
7  }
```

Transactions with repositories?

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val u = User(null, "me@asm0dey.site", 37)
4      val savedId = repo.save(u)
5      repo.save(u.copy(id = savedId)) // will throw
6      return curId
7  }
```

Transactions with repositories?

```
1  @Transactional
2  suspend fun failTransactional(): Long {
3      val u = User(null, "me@asm0dey.site", 37)
4      val savedId = repo.save(u)
5      repo.save(u.copy(id = savedId)) // will throw
6      return curId
7  }
```

Configuration

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Modified Jackson's `ObjectMapper` to work with `data` classes from `jackson-module-kotlin`

```
1 @Bean  
2 fun kotlinMapper(): ObjectMapper {  
3     return jacksonObjectMapper()  
4 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Modified Jackson's `ObjectMapper` to work with `data` classes from `jackson-module-kotlin`

```
1 @Bean  
2 fun kotlinMapper(): ObjectMapper {  
3     return jacksonObjectMapper()  
4 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Modified Jackson's `ObjectMapper` to work with `data` classes from `jackson-module-kotlin`

```
1 @Bean  
2 fun kotlinMapper(): ObjectMapper {  
3     return jacksonObjectMapper()  
4 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Modified Jackson's `ObjectMapper` to work with `data` classes from `jackson-module-kotlin`

```
1 @Bean  
2 fun kotlinMapper(): ObjectMapper {  
3     return jacksonObjectMapper()  
4 }
```

Let's start simple

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3 }
```

Modified Jackson's `ObjectMapper` to work with `data` classes from `jackson-module-kotlin`

```
1 @Bean  
2 fun kotlinMapper(): ObjectMapper {  
3     return jacksonObjectMapper()  
4 }
```

4 lines instead of 1 😱

Custom bean

```
1  class JsonLogger(private val objectMapper: ObjectMapper) {
2      fun log(o: Any) {
3          if (o::class.isData) {
4              println(objectMapper.writeValueAsString(o))
5          } else println(o.toString())
6      }
7  }
8
9  val beans = beans {
10     bean { jacksonObjectMapper() }
11     bean(::JsonLogger)
12 }
```

Custom bean

```
1  class JsonLogger(private val objectMapper: ObjectMapper) {
2      fun log(o: Any) {
3          if (o::class.isData) {
4              println(objectMapper.writeValueAsString(o))
5          } else println(o.toString())
6      }
7  }
8
9  val beans = beans {
10     bean { jacksonObjectMapper() }
11     bean(::JsonLogger)
12 }
```

Custom bean

```
1  class JsonLogger(private val objectMapper: ObjectMapper) {
2      fun log(o: Any) {
3          if (o::class.isData) {
4              println(objectMapper.writeValueAsString(o))
5          } else println(o.toString())
6      }
7  }
8
9  val beans = beans {
10     bean { jacksonObjectMapper() }
11     bean(::JsonLogger)
12 }
```

Custom bean

```
1  class JsonLogger(private val objectMapper: ObjectMapper) {
2      fun log(o: Any) {
3          if (o::class.isData) {
4              println(objectMapper.writeValueAsString(o))
5          } else println(o.toString())
6      }
7  }
8
9  val beans = beans {
10     bean { jacksonObjectMapper() }
11     bean(::JsonLogger)
12 }
```

Custom bean

```
1  class JsonLogger(private val objectMapper: ObjectMapper) {
2      fun log(o: Any) {
3          if (o::class.isData) {
4              println(objectMapper.writeValueAsString(o))
5          } else println(o.toString())
6      }
7  }
8
9  val beans = beans {
10     bean { jacksonObjectMapper() }
11     bean(::JsonLogger)
12 }
```

Arbitrary logic

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3     bean(::JsonLogger)  
4     bean("randomGoodThing", isLazyInit = Random.nextBoolean()) {  
5         if (Random.nextBoolean()) "Norway" else "Well"  
6     }  
7 }
```

Arbitrary logic

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3     bean(::JsonLogger)  
4     bean("randomGoodThing", isLazyInit = Random.nextBoolean()) {  
5         if (Random.nextBoolean()) "Norway" else "Well"  
6     }  
7 }
```

Arbitrary logic

```
1 val beans = beans {  
2     bean { jacksonObjectMapper() }  
3     bean(::JsonLogger)  
4     bean("randomGoodThing", isLazyInit = Random.nextBoolean()) {  
5         if (Random.nextBoolean()) "Norway" else "Well"  
6     }  
7 }
```

OK How do I use it?

Let's return to our very first file

```
1 runApplication<SampleApplication>(*args)
```

```
1 val beans = { /* */ }
```

OK How do I use it?

Let's return to our very first file

```
1 runApplication<SampleApplication>(*args)
```

```
1 val beans = { /* */ }
```

Let's change it to

```
1 runApplication<SampleApplication>(*args) {
2     addInitializers(beans)
3 }
```

OK How do I use it?

Let's return to our very first file

```
1 runApplication<SampleApplication>(*args)
```

```
1 val beans = { /* */ }
```

Let's change it to

```
1 runApplication<SampleApplication>(*args) {  
2     addInitializers(beans)  
3 }
```

OK How do I use it?

Let's return to our very first file

```
1 runApplication<SampleApplication>(*args)
```

```
1 val beans = { /* */ }
```

Let's change it to

```
1 runApplication<SampleApplication>(*args) {  
2     addInitializers(beans)  
3 }
```

And run it...

```
1 Started SampleApplicationKt in 1.776 seconds (process running for 2.133)
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Let's test it

Bean:

```
1  @Component
2  class MyBean(val jsonLogger: JsonLogger) {
3      fun test() = jsonLogger.log("Test")
4  }
```

Test:

```
1  @SpringBootTest
2  class ConfigTest {
3      @Autowired private lateinit var myBean: MyBean
4      @Test
5      fun testIt() = assertEquals("Test", myBean.test())
6  }
```

Run it

```
No qualifying bean of type  
'com.github.asm0dey.sample.JsonLogger'  
available: expected at least 1  
bean which qualifies as autowire candidate
```

Run it

```
No qualifying bean of type  
'com.github.asm0dey.sample.JsonLogger'  
available: expected at least 1  
bean which qualifies as autowire candidate
```



Run it

```
No qualifying bean of type  
'com.github.asm0dey.sample.JsonLogger'  
available: expected at least 1  
bean which qualifies as autowire candidate
```

That's because our tests do not call
main !



Requires some glue to work

```
1 val beans = { /* */ }
2 class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3     override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4 }
```

Requires some glue to work

```
1 val beans = { /* */ }
2 class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3     override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4 }
```

Requires some glue to work

```
1  val beans = { /* */ }
2  class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3      override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4  }
```

Requires some glue to work

```
1  val beans = { /* */ }
2  class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3      override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4  }
```

Requires some glue to work

```
1 val beans = { /* */ }
2 class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3     override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4 }
```

application.yml :

```
1 context.initializer.classes: "com.github.asm0dey.sample.BeansInitializer"
```

Requires some glue to work

```
1 val beans = { /* */ }
2 class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3     override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4 }
```

application.yml :

```
1 context.initializer.classes: "com.github.asm0dey.sample.BeansInitializer"
```

Main.kt :

```
1 fun main(args: Array<String>) {
2     runApplication<SampleApplication>(*args)
3 }
```

Requires some glue to work

```
1 val beans = { /* */ }
2 class BeansInitializer : ApplicationContextInitializer<GenericApplicationContext> {
3     override fun initialize(context: GenericApplicationContext) = beans.initialize(context)
4 }
```

application.yml :

```
1 context.initializer.classes: "com.github.asm0dey.sample.BeansInitializer"
```

Main.kt :

```
1 fun main(args: Array<String>) {
2     runApplication<SampleApplication>(*args)
3 }
```

Security

Spring Security

So, what did I learn?

So, what did I learn?

So, what did I learn?

- Always generate the project with start.spring.io

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives
- `data` classes should not be used for JPA

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives
- `data` classes should not be used for JPA
- JDBC is simpler with Kotlin

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives
- `data` classes should not be used for JPA
- JDBC is simpler with Kotlin
- R2DBC is simpler with coroutines

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives
- `data` classes should not be used for JPA
- JDBC is simpler with Kotlin
- R2DBC is simpler with coroutines
- Bean definition DSL is awesome

So, what did I learn?

- Always generate the project with start.spring.io
- Reified generics might make an API better
- Validation is better with Kotlin, but remember about primitives
- `data` classes should not be used for JPA
- JDBC is simpler with Kotlin
- R2DBC is simpler with coroutines
- Bean definition DSL is awesome
- Specifically with security!

Thank you!

Thank you! Questions?

-  asm0di0
-  @asm0dey@fosstodon.org
-  me@asm0dey.site
-  asm0dey
-  asm0dey
-  asm0dey
-  asm0dey
-  asm0dey



END