TODO App Backend

Database Auditing

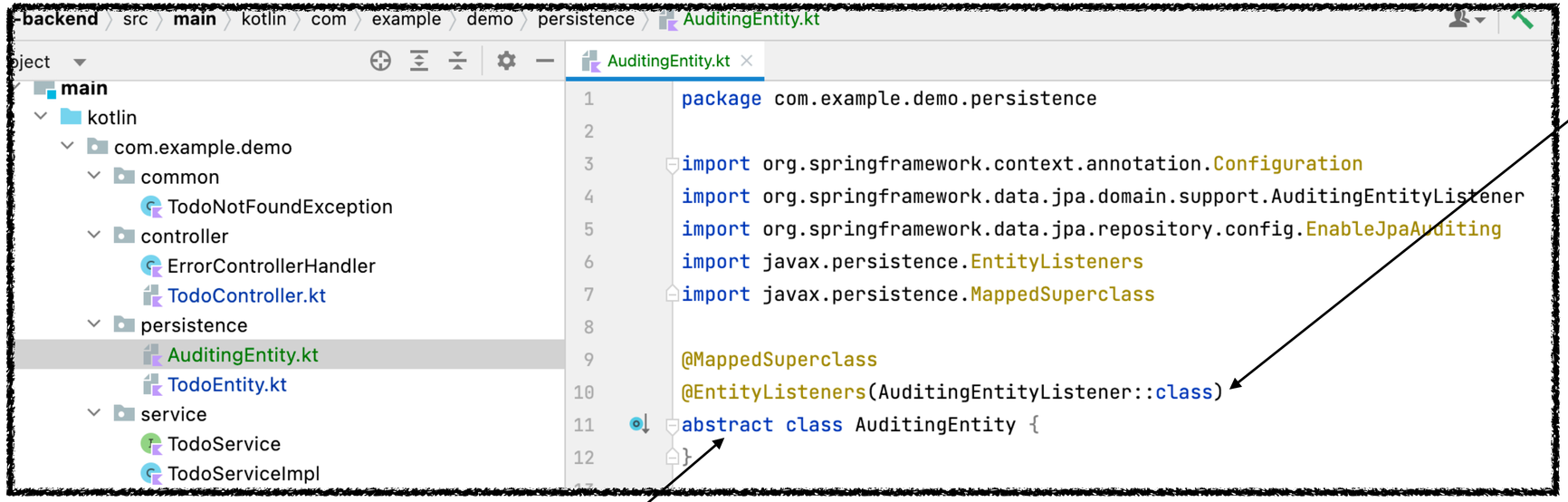
Exercise 8

- database auditing means tracking and logging events related to persistent entities, or simply entity versioning
- Inspired by SQL triggers, the events are insert, update, and delete operations on entities
- The benefits of database auditing are analogous to those provided by source version control

Add an Auditing Entity / Base Entity

Exercise 8

JPA provides the @EntityListeners annotation to specify callback listener classes. Spring Data provides its own JPA entity listener class, AuditingEntityListener.
-
Now we can capture auditing information by the listener upon persisting and updating an entity That inherits our BaseEntity

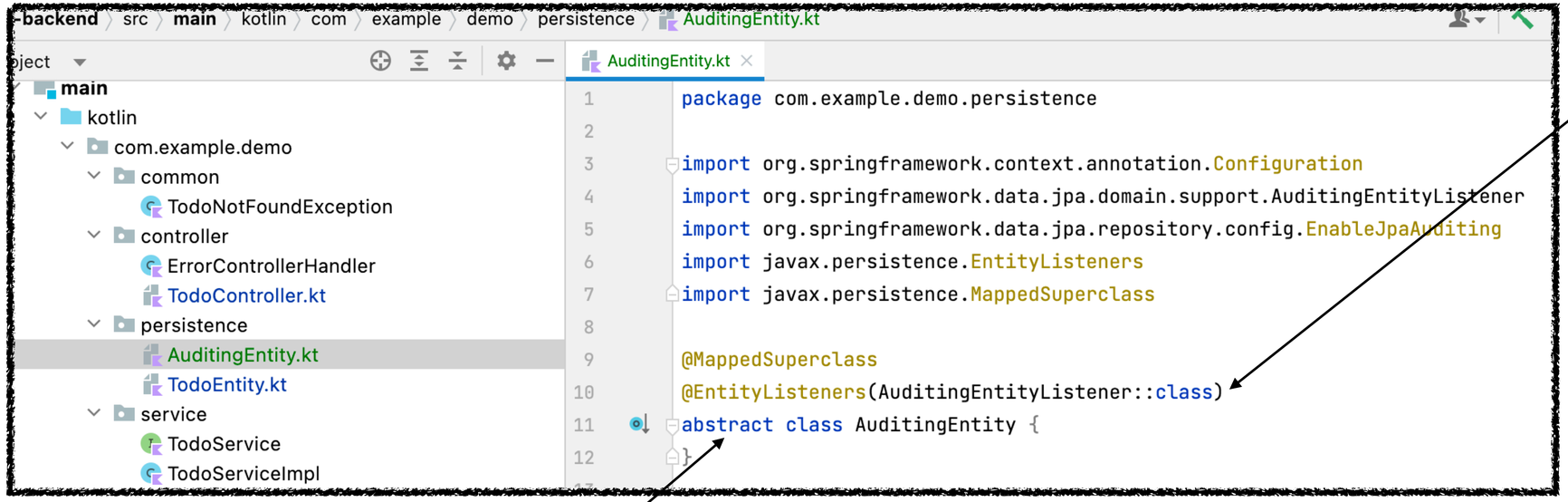


Little trick: by making our AuditingEntity class abstract we prevent others to instantiate this class as an object (not possible for abstract classes). if one wants to „use“ it, it needs to be inherited.

Add a Base Entity

Exercise 8

JPA provides the @EntityListeners annotation to specify callback listener classes. Spring Data provides its own JPA entity listener class, AuditingEntityListener.
-
Now we can capture auditing information by the listener upon persisting and updating an entity That inherits our BaseEntity



Little trick: by making our AuditingEntity class abstract we prevent others to instantiate this class as an object (not possible for abstract classes). if one wants to „use“ it, it needs to be inherited.

Enable Database Auditing with Spring Data JPA

Exercise 8

```
@MappedSuperclass
@EntityListeners(AuditingEntityListener::class)
abstract class AuditingEntity {
}

@Configuration
@EnableJpaAuditing
class PersistenceConfig
```

Add a configuration bean that enables JPA auditing. We can just do that in our AuditingEntity file

Track Created and Last Modified Dates

Exercise 8

```
@MappedSuperclass
@EntityListeners(AuditingEntityListener::class)
abstract class AuditingEntity {

    @Column(name = "created_date", nullable = false, updatable = false)
    @CreatedDate
    @Temporal(TemporalType.TIMESTAMP)
    var createdDate: Date = Date()

    @Column(name = "modified_date")
    @LastModifiedDate
    @Temporal(TemporalType.TIMESTAMP)
    var modifiedDate: Date = Date()
}

@Configuration
@EnableJpaAuditing
class PersistenceConfig
```

add two new properties for storing the created and last modified dates to our Bar entity. The properties are annotated by the @CreatedDate and @LastModifiedDate annotations accordingly, and their values are set automatically

Use the AuditingEntity

Exercise 8

```
@Table(name = "todo")
@Entity(name = "todo")
class TodoEntity(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Int = 0,

    @Column(name = "task")
    var task: String,

    var completed: Boolean,
): AuditingEntity()
```

We use our auditing base entity by inheriting it to other entities

Pass information to controllers response

Exercise 8

TodoController.kt

```
68  ) {
69      fun toEntity() : TodoEntity = TodoEntity(
70          task = task,
71          completed = completed
72      )
73  }
74
75  data class TodoResponse(
76      val id: Int,
77      val task: String,
78      val completed: Boolean,
79      val created: Date,
80      val lastUpdated: Date
81  )
82
83  fun TodoEntity.toResponse() : TodoResponse = TodoResponse(id, task, completed, createdDate, modifiedDate)
84  fun List<TodoEntity>.toResponse() : List<TodoResponse> = map { it.toResponse() }
85
```

Add fields to response

Add fields to mapping

Pass information to controllers response

Exercise 8

TodoController.kt

```
68  ) {
69      fun toEntity() : TodoEntity = TodoEntity(
70          task = task,
71          completed = completed
72      )
73  }
74
75  data class TodoResponse(
76      val id: Int,
77      val task: String,
78      val completed: Boolean,
79      val created: Date,
80      val lastUpdated: Date
81  )
82
83  fun TodoEntity.toResponse() : TodoResponse = TodoResponse(id, task, completed, createdDate, modifiedDate)
84  fun List<TodoEntity>.toResponse() : List<TodoResponse> = map { it.toResponse() }
85
```

Add fields to response

Add fields to mapping

Verify

Exercise 8

GET http://localhost:8080/todo/all Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Body Cookies Headers (5) Test Results 200 OK 93 ms 299 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "task": "task 1337",
        "completed": false,
        "created": "2022-04-20T19:40:19.403+00:00",
        "lastUpdated": "2022-04-20T19:40:25.681+00:00"
    }
]
```


When creating / updating a todo
the corresponding fields of our auditing entity
will be updated

Need more custom callbacks on persist actions?

Exercise 8

AuditingEntity.kt

```kotlin
@MappedSuperclass
@EntityListeners(AuditingEntityListener::class)
abstract class AuditingEntity {

    @Column(name = "created_date", nullable = false, updatable = false)
    @CreatedDate
    @Temporal(TemporalType.TIMESTAMP)
    var createdDate: Date = Date()

    @Column(name = "modified_date")
    @LastModifiedDate
    @Temporal(TemporalType.TIMESTAMP)
    var modifiedDate: Date = Date()

    @PrePersist
    fun onPrePersist() {
        println("this method will be executed everytime before this entity will be persisted")
    }

    @PreUpdate
    fun onPreUpdate() {
        println("this method will be executed everytime before this entity will be updated")
    }

    @PreRemove
    fun onPreRemove() {
        println("this method will be executed everytime before this entity will be deleted")
    }
}
```

In a JPA Entity class, we can specify a method as a callback, which we can invoke during a particular entity lifecycle event.

-

The functions name can be whatever you want