ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Spring Framework

Αθ. Ανδρούτσος

Spring Data

- Τα **Spring Data** είναι εκείνο το Spring module που διαχειρίζεται το θέμα του *persistence*
- Το κεντρικό **interface** στο Spring Data είναι το ***Repository***
- Η βασική ιδέα είναι ότι **υπάρχει μία ιεραρχία από άλλα interfaces κάτω από το *Repository* που το κάνουν extends** και έχουμε άλλα sub-repositories που το καθένα διαχειρίζεται μία διαφορετική τεχνολογία persistence, π.χ. JDBC, JPA, NoSQL, κλπ.

Spring Data Repositories (2)

- Μέσω του interface **Repository** και των υπόλοιπων sub-repositories το Spring παρέχει DAO-Layer services και έτσι εμείς δεν χρειάζεται να υλοποιήσουμε το DAO Layer
- **Αρκεί για κάθε domain class να ορίζουμε ένα repository** που να κάνει extends ένα συγκεκριμένο sub-repository (π.χ. ***JpaRepositiry***) ανάλογα με την τεχνολογία persistence που θέλουμε να χρησιμοποιήσουμε
- Στη συνέχεια το Spring υλοποιεί αυτόματα τις μεθόδους που ορίζονται στο interface καθώς και άλλες CRUD μεθόδους και εμείς δεν χρειάζεται να κάνουμε τίποτα άλλο εκτός αν θέλουμε κάποια custom queries

Repository

docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html?is-external=true

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.springframework.data.repository

Interface Repository<T,ID>

Type Parameters:

T - the domain type the repository manages

ID - the type of the id of the entity the repository manages

All Known Subinterfaces:

CrudRepository<T,ID>, PagingAndSortingRepository<T,ID>, ReactiveCrudRepository<T,ID>, ReactiveSortingRepository<T,ID>, RevisionRepository<T,ID,N>, RxJava2CrudRepository<T,ID>, RxJava2SortingRepository<T,ID>, RxJava3CrudRepository<T,ID>, RxJava3SortingRepository<T,ID>

```
@Indexed
public interface Repository<T,ID>
```

Central repository marker interface. Captures the domain type to manage as well as the domain type's id type. General purpose is to hold type information as well as being able to discover interfaces that extend this one during classpath scanning for easy Spring bean creation.

Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the same signature as those declared in CrudRepository.

Author:

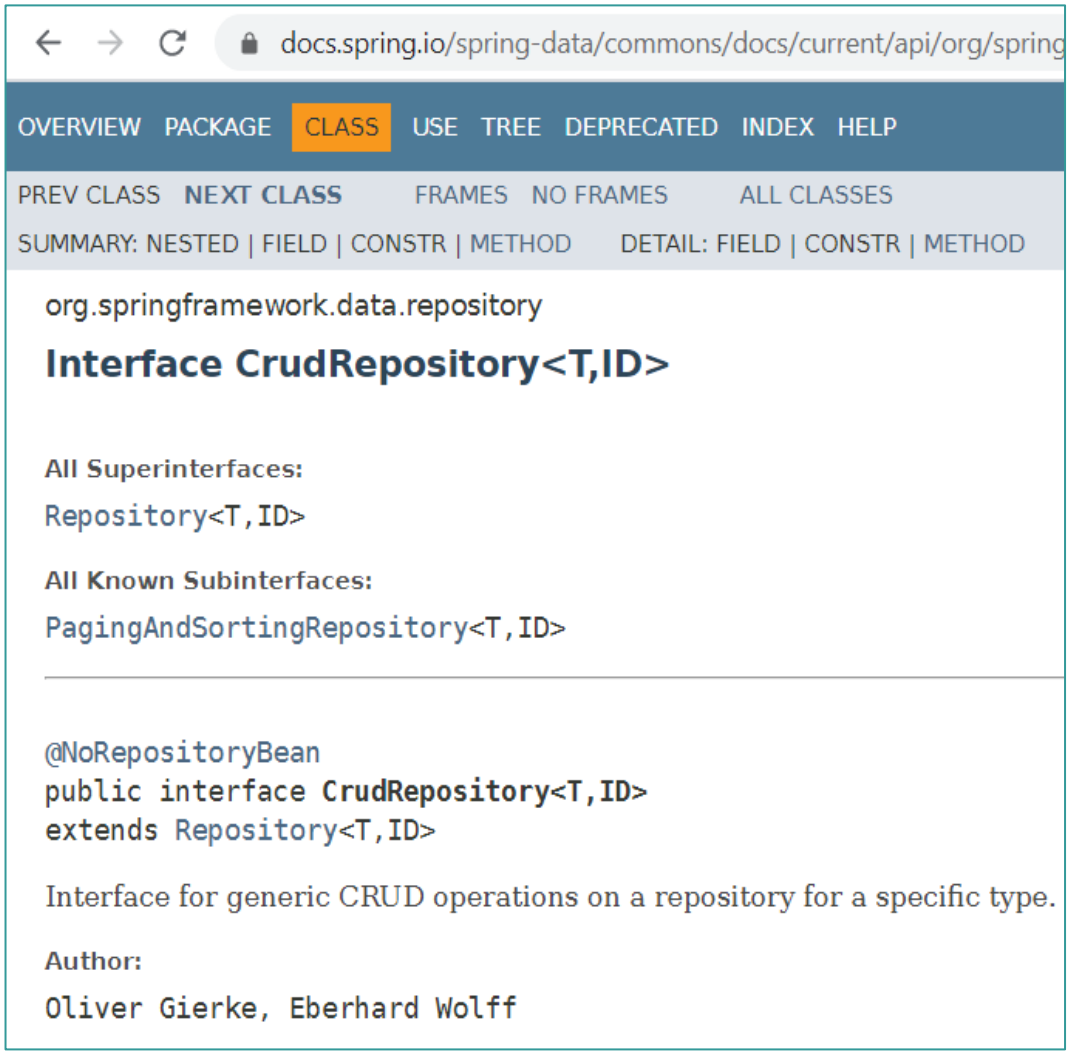
Oliver Gierke

See Also:

CrudRepository

- Το interface Repository<T, id> είναι ένα **marker interface** και δεν ορίζει καμία μέθοδο. Λαμβάνει δύο παραμετρικούς **Generic τύπους, T και ID**, όπου **T είναι ο τύπος της domain κλάσης που θέλουμε να διαχειρίζεται το Repository** και το **ID είναι ο τύπος του Id του Entity** που το κάνει identifiable. Για παράδειγμα θα μπορούσε να είναι <Student, Long>

CrudRepository (1)



- Ένα σημαντικό interface που κάνει extends το Repository είναι το **CrudRepository**
- Μας παρέχει βασικές persistence πράξεις, όπως
  - **save**(S entity), που **αποθηκεύει** ένα Entity αν δεν υπάρχει (empty id) ή το κάνει **update** (merge) αν υπάρχει (non-empty id)
  - **delete**(T entity) που διαγράφει ένα Entity
  - **deleteById(ID id)**, που διαγράφει με βάση το id του Entity
  - **Optional<T> findById(ID id)** retrieves ένα entity με βάση το id του Entity
  - **boolean existsById(ID id),** επιστρέφει αν ένα entity με το id exists
- Στην επόμενη διαφάνεια παρουσιάζονται όλες οι πράξεις του CrudRepository

CrudRepository (2)

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
long	count()	Returns the number of entities available.
void	delete(T entity)	Deletes a given entity.
void	deleteAll()	Deletes all entities managed by the repository.
void	deleteAll(Iterable <? extends T> entities)	Deletes the given entities.
void	deleteAllById(Iterable <? extends ID> ids)	Deletes all instances of the type T with the given IDs.
void	deleteById(ID id)	Deletes the entity with the given id.
boolean	existsById(ID id)	Returns whether an entity with the given id exists.
Iterable <T>	findAll()	Returns all instances of the type.
Iterable <T>	findAllById(Iterable <ID> ids)	Returns all instances of the type T with the given IDs.
Optional <T>	findById(ID id)	Retrieves an entity by its id.
<S extends T> S	save(S entity)	Saves a given entity.
<S extends T> Iterable <S>	saveAll(Iterable <S> entities)	Saves all given entities.

- Πρόκειται για 12 μεθόδους που παρέχουν CRUD υπηρεσίες

PagingAndSortingRepository

docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.ht

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.springframework.data.repository

Interface PagingAndSortingRepository<T,ID>

All Superinterfaces:

CrudRepository<T,ID>, Repository<T,ID>

```
@NoRepositoryBean
public interface PagingAndSortingRepository<T,ID>
extends CrudRepository<T,ID>
```

Extension of CrudRepository to provide additional methods to retrieve entities using the pagination and sorting abstraction.

Author:

Oliver Gierke

See Also:

Sort, Pageable, Page

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
Page<T>	findAll(Pageable pageable) Returns a Page of entities meeting the paging restriction provided in the Pageable object.
Iterable<T>	findAll(Sort sort) Returns all entities sorted by the given options.

- Κάνει extends το *Repository* και ορίζει δύο επιπλέον μεθόδους για να μπορούμε να έχουμε access σε pages ενός μεγάλου query και να το παρουσιάζουμε τμηματικά στους χρήστες καθώς και sortαρισμένα
- Συνδυάζεται με το *CrudRepository* για να παρέχουμε και CRUD functionality

JpaRepository (1)

docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.springframework.data.jpa.repository

Interface JpaRepository<T,ID>

All Superinterfaces:

CrudRepository<T,ID>, PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>, Repository<T,ID>

All Known Subinterfaces:

JpaRepositoryImplementation<T,ID>

All Known Implementing Classes:

QuerydslJpaRepository, SimpleJpaRepository

```
@NoRepositoryBean
public interface JpaRepository<T,ID>
extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>
```

JPA specific extension of Repository.

Author:

Oliver Gierke, Christoph Strobl, Mark Paluch

- Υλοποιεί το **JPA**. Ο default implementation provider είναι το Hibernate. Κάνει extends το *CrudRepository* και το *PagingAndSortingRepository* και παρέχει επιπλέον μεθόδους

JpaRepository (2)

All Methods | Instance Methods | Abstract Methods | Default Methods | Deprecated Methods

Modifier and Type	Method	Description
void	`deleteAllByIdInBatch(Iterable <ID> ids)`	Deletes the entities identified by the given ids using a single query.
void	`deleteAllInBatch()`	Deletes all entities in a batch call.
void	`deleteAllInBatch(Iterable <T> entities)`	Deletes the given entities in a batch which means it will create a single query.
default void	`deleteInBatch(Iterable <T> entities)`	**Deprecated.** Use `deleteAllInBatch(Iterable)` instead.
`<S extends T> List <S>`	`findAll(Example <S> example)`	
`<S extends T> List <S>`	`findAll(Example <S> example, Sort sort)`	
void	`flush()`	Flushes all pending changes to the database.
T	`getById(ID id)`	**Deprecated.** use `getReferenceById(ID)` instead.
T	`getOne(ID id)`	**Deprecated.** use `getReferenceById(ID)` instead.
T	`getReferenceById(ID id)`	Returns a reference to the entity with the given identifier.
`<S extends T> List <S>`	`saveAllAndFlush(Iterable <S> entities)`	Saves all entities and flushes changes instantly.
`<S extends T> S`	`saveAndFlush(S entity)`	Saves an entity and flushes changes instantly.

- Παρέχονται μέθοδοι, όπως **T getReferenceById(ID id)** (οι getById και getOne είναι deprecated) που είναι παρόμοια με την *findById* του *CrudRepository* αλλά επιστρέφει entity αντί για Optional καθώς και exception αν δεν βρεθεί το entity. Επομένως είναι προτιμότερη η findById του CRUD Repository

Spring Data JPA

- Το Spring Data JPA είναι μέρος του Spring Data για να υλοποιούμε JPA repositories
- Το εισάγουμε με spring-boot-starter-data-jpa του Spring boot που μπορούμε να κάνουμε include στο Gradle/Maven. Ο **default JPA provider είναι το Hibernate** και το **default connection pool είναι το HikariCP**

```
15  dependencies {
16      implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
17      implementation 'org.springframework.boot:spring-boot-starter-web'
18      implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
19
20      compileOnly 'org.projectlombok:lombok'
21      annotationProcessor 'org.projectlombok:lombok'
22      testImplementation 'org.springframework.boot:spring-boot-starter-test'
23      runtimeOnly 'com.mysql:mysql-connector-j'
```

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.4.5</version>
</dependency>
```

Spring Data JPA - JpaRepository

- Όλες οι μέθοδοι του **JpaRepository** υλοποιούνται αυτόματα από το Spring χωρίς εμείς να χρειάζεται να υλοποιήσουμε τίποτα, απλά ορίζουμε ένα δικό μας Repository με το annotation **@Repository** για κάθε Entity που θέλουμε να διαχειριστούμε και κάνουμε extends το **JpaRepository**
- Για παράδειγμα ορίζουμε παρακάτω το **StudentRepository**:

```
2  import org.springframework.data.jpa.repository.JpaRepository;
3  import org.springframework.stereotype.Repository;
4
5  @Repository
6  public interface StudentRepository extends JpaRepository<Student, Long> {
7  }
```

Custom Methods

- Αν θέλουμε ***custom queries***, υπάρχουν δύο τρόποι:
  - Να τα δηλώσουμε -μέσα στο δικό μας Repository- με βάση κάποια **naming conventions (Derived Queries)**
  - Να χρησιμοποιήσουμε **custom queries (με @Query και JPQL)**

Derived Queries (1)

- Τα Spring Data derived queries είναι ένα χαρακτηριστικό του Spring Data JPA που επιτρέπει στους προγραμματιστές να **ορίζουν ερωτήματα database queries απλώς δηλώνοντας ονόματα μεθόδων** στα Repository interfaces
- Το Spring Data JPA παρέχει naming conventions για τα derived queries με βάση **ρήματα** με τα οποία ξεκινά το όνομα μίας μεθόδου :
  - **find…By**
  - **get…By,**
  - **read…By,**
  - **query…By** που είναι ισοδύναμα και επιστέφουν objects καθώς και το
  - **count…By** που επιστέφει το πλήθος των objects

Derived Queries (2)

- Μετά το get, read, find ή το count μπορεί να εισαχθεί (δεν είναι απαραίτητο) μία λέξη όπως το object name που επιστέφουμε ή κάποια άλλη λέξη και στη συνέχεια το By και μετά το Predicate που μπορεί να είναι μία ή περισσότερες ιδιότητες
- Υπάρχουν λοιπόν, σε ένα Derived Query, δύο μέρη που χωρίζονται με το **By**: 1) *introducer / action* (get, find, read, query, count) και 2) *criteria* όπως ByLastname για το πεδίο lastname
- Κατ' αυτό τον τρόπο με τα naming conventions μπορούμε εύκολα να κάνουμε derive ένα query, απλά κοιτώντας το όνομά του

Derived Queries (3)

- Για παράδειγμα, αν θέλουμε να ορίσουμε μία μέθοδο που να επιστέφει Teachers με βάση την ιδιότητα **lastname του Entity Teacher**, τότε το όνομα της μεθόδου θα μπορούσε να είναι ***findByLastname()*** που μπορεί να επιστρέφει List<Teacher>

```
1  public interface TeacherRepository extends JpaRepository<Teacher, Long> {
2      List<Teacher> findByLastname(String lastname);
3      Page<Teacher> findByCity(String city, Pageable pageable);
4      Optional<Teacher> findByVat(String vat)
5  }
```

Derived Queries (4)

- Η λέξη Teachers ή οτιδήποτε εισάγουμε πριν το By και μετά το find/get/… δεν είναι υποχρεωτικό (μιας και το Spring δεν κάνει έλεγχο) παρά μόνο αν έχουμε τα **Distinct, First ή Top** για να κάνουμε remove τα duplicates ή να περιορίσουμε τα results αντίστοιχα (π.χ. findDistinctByLastname)
- Οπότε το query που δημιουργείται αυτόματα πρέπει να ξέρει ότι επιστρέφεται μοναδικό result set ή το πλήθος των First ή Top instances (π.χ. findTop3ByFirstname())

Derived Queries (5)

- Οι συνθήκες στο Predicate μπορούν να συνενώνονται με And ή Or όπως: *List<Student> findByFirstnameOrLastname (String first, String last)*
  List<Student> findByFirstnameAndLastname (String first, String last)

```
IsAfter, After, IsGreaterThan, GreaterThan
IsGreaterThanEqual, GreaterThanEqual
IsBefore, Before, IsLessThan, LessThan
IsLessThanEqual, LessThanEqual
IsBetween, Between
IsNull, Null
IsNotNull, NotNull
IsIn, In
IsNotIn, NotIn
IsStartingWith, StartingWith, StartsWith
IsEndingWith, EndingWith, EndsWith
IsContaining, Containing, Contains
IsLike, Like
IsNotLike, NotLike
IsTrue, True
IsFalse, False
Is, Equals
IsNot, Not
```

Επίσης μπορούμε να χρησιμοποιήσουμε και **comparison operators**

Για να ταξινομήσουμε χρησιμοποιούμε το **OrderBy**. Για παράδειγμα:

List<Student> getByFirstnameOrderByFirstnameAsc(String firstname);

List<Student> getByLastnameOrderByFirstnameDesc(String last);

Το StartingWith γίνεται translate σε LIKE 'value%', π.χ. findByLastnameStartingWith(String lastname)

Derived Queries (6)

Boolean Keyword	Παράδειγμα	JPQL
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is **Equals**	findByFirstnameIs findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null

Derived Queries (7)

Boolean Keyword	Παράδειγμα	JPQL
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1 (not wrapped in %)
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1(with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1(with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1(wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstname) = UPPER(?1)

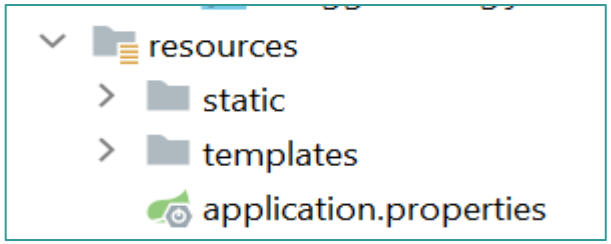
Custom queries

- Εκτός από τα derived queries μπορούμε να δημιουργούμε custom queries με το **@Query** annotation και μπορούμε να ορίσουμε *JPQL* ή *native SQL queries.* Αυτή η δυνατότητα χρειάζεται όταν έχουμε σύνθετα queries με JOIN. Τα derived queries δουλεύουν σε single entities.
- Έστω ότι θέλουμε να ελέγξουμε αν το username και το password ενός χρήστη υπάρχει στη ΒΔ

```java
package gr.aueb.cf.springhello.repository;

import gr.aueb.cf.springhello.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsernameEquals(String username);

    @Query("SELECT count(*) > 0 FROM User U WHERE U.username = ?1 AND U.password = ?2")
    boolean isUserValid(String username, String password);
}
```

application.properties (1)

- Για να χρησιμοποιήσουμε μία ΒΔ θα πρέπει να ορίσουμε στο αρχείο **application.properties** τις ιδιότητες του data-source, με παρόμοιο τρόπο όπως είχαμε δει στο Hibernate (όσο για το Dialect προσδιορίζεται αυτόματα ποιο dialect να χρησιμοποιήσει, από τον Connector)



Επίσης, θα πρέπει να συμπεριλάβουμε στα Dependencies του Gradle/Maven τον driver της ΒΔ

```
spring.datasource.url=jdbc:mysql://localhost:3306/tsdb22?serverTimezone=UTC
spring.datasource.username=thanos22
spring.datasource.password=Thanos22
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

```
23        runtimeOnly 'com.mysql:mysql-connector-j'
```

application.properties (2)


```
spring.datasource.url=jdbc:mysql://localhost:3306/tsdb22?serverTimezone=UTC
spring.datasource.username=thanos22
spring.datasource.password=Thanos22
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

- Στο application.properties ορίζουμε το connection string και το driver class
- Το dialect γίνεται detect αυτόματα
- Επίσης, αυτόματα παρέχεται το Hikari connection pool με sensible defaults (αν θέλουμε κάνουμε overwrite)

application.properties (3)

- Στην ανάπτυξη εφαρμογών έχουμε τρία βασικά περιβάλλοντα ανάπτυξης: 1) test (συνήθως προσωπικό pc), 2) staging (σαν το παραγωγικό αλλά το δίνουμε για να πάρουμε feedback) και, 3) το παραγωγικό
- Μπορούμε στο application.properties να ορίσουμε τα βασικά properties και τα properties που αφορούν το connection να πάνε σε application-*.properties αρχεία που αντιστοιχούν στα τρία παραπάνω περιβάλλοντα τα οποία έχουν διαφορετικές ΒΔ
- Θα το δούμε στο project που θα αναπτύξουμε

Validation (1)

- Το validation αφορά την **επαλήθευση του input του χρήστη**, ώστε να μην φτάσουν data στα services και στη συνέχεια στην ΒΔ που να μην είναι ορθά ως προς τον *τύπο* τους, τα *μέγεθός* τους και το *range* που θα θέλαμε αλλά και ως προς τη *λογική* τους. Τυπικά το Validation γίνεται στον Controller
- Στο *build.gradle* εισάγουμε

```
implementation 'org.springframework.boot:spring-boot-starter-validation'
```

Validation στον client

- Στον client μπορεί να γίνει validation με HTML Validations και JavaScript, αλλά δεν μπορούμε να βασιστούμε για το validation σε μηχανήματα που δεν ελέγχουμε και όπου μπορεί ο έλεγχος να παρακαμφθεί
- Ο βασικός λόγος για validation στον client είναι για βελτίωση του User Experience μιας και ο έλεγχος γίνεται γρήγορα και ανά πεδίο της φόρμας

Validation (2)

- Το Spring παρέχει δύο μηχανισμούς για Validation:
  - **Bean Validation**. Αφορά το *validation ενός Java Bean* με τη χρήση annotations (@NotNull, @Min, @Max, @Size, @Pattern, @Email) στα πεδία του Bean. Το Spring bean validation γίνεται by default από τον **Hibernate Validator**
  - **Custom Validation.** Μπορεί και εδώ να γίνει validation με Java. Επίσης, επιπλέον μπορεί να γίνει **validation ως προς τη λογική της εφαρμογής** μας. Δηλαδή μπορεί μεν να φτάσει ένα έγκυρο Date για booking σε ένα booking service αλλά το service να ελέγξει αν κάνει overlap με ημερομηνία που είναι ήδη booked, οπότε δεν προχωρά να κάνει πράξεις στη ΒΔ.

Bean Validation

- Το Bean Validation (Bean Validation 2.0, JSR 380) είναι μέρος του Java API και του Java EE (*jakarta.validation*)
- Στο πλαίσιο εφαρμογών, Bean Validation μπορεί να χρησιμοποιηθεί στα DTOs
- Το bean validation μπορεί να χρησιμοποιηθεί και από τον Swagger για να τεκμηριώσει τα Schemas (DTOs)

Bean validation - Παράδειγμα


```java
package gr.aueb.cf.datatest.dto;

import ...

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class TeacherInsertDTO {

    @NotNull(message = "isActive field is required")
    private Boolean isActive;

    @NotNull(message = "User details are required")
    private UserInsertDTO user;

    @NotNull(message = "Personal Info is required")
    private PersonalInfoInsertDTO personalInfo;
}
```

- Είναι όπως το έχουμε δει στο Jakarta / Hibernate validation

Custom Validation

- Το Custom validation αφορά όλη την εφαρμογή και τυπικά υλοποιείται σε ένα ξεχωριστό Layer, το **validator** Layer (π.χ. *validator* package)
- Είναι δηλαδή ένα Layer που παρέχει **υπηρεσίες Validation οριζόντια στην εφαρμογή μας**
- **Custom validation** γίνεται κάνοντας implement το interface ***org.springframework.validation.Validator*** του Spring και υλοποιώντας τις δύο μεθόδους του: ***supports* και *validate***
- Μπορούμε να έχουμε τόσους custom validators όσα backing-beans (και αντίστοιχες φόρμες) θέλουμε να κάνουμε validate

Spring's Validator interface

- Το interface *org.springframework.validation.**Validator*** το χρησιμοποιούμε για να κάνουμε validate objects
- Χρησιμοποιείται μαζί με την κλάση **Errors** ώστε καθώς κάνει validate το object, τυχόν λάθη στο validation, να εισάγονται στο **errors object**
- Μέθοδοι:
  - boolean **supports**(Class clazz) – όπου clazz είναι τα instanxces ποιας κλάσης κάνουμε validate
  - void **validate**(Object obj, Errors errors) – κάνει validates το object obj και σε περίπτωση validation errors τα κάνει register στο errors object
- Παρέχεται επίσης από το Spring η **ValidationUtils** helper class που ελέγχει πεδία της κλάσης

Controller

- Ο **Controller** είναι το επίπεδο κλήσης υπηρεσιών Validation ώστε να διασφαλιστεί το ορθό format των δεδομένων εισόδου από τον χρήστη αλλά και η λογική της εφαρμογής
- Μπορεί ο **Controller να καλεί υπηρεσίες Validation** τόσο του custom Validator όσο και του Bean Validator
- Ωστόσο, ο μηχανισμός που κάνουμε register τα errors είναι ίδιος

Validation στον Controller

- Ο Controller μπορεί να καλέσει υπηρεσίες validation ως εξής:
  - **Bean Validation**. Με το annotation **@Valid** στο backing bean. Τυχόν λάθη γίνονται αυτόματα register στο interface **BindingResult** που *είναι στις τυπικές παραμέτρους* της μεθόδου
  - Καλώντας την μέθοδο **validate** του Validator (*org.springframework.validation.Validator)* που έχουμε υλοποιήσει. Κι εδώ τα errors γίνονται register στο BindingResult (στη μέθοδο validate περνάμε ως παράμετρο το **bindingResult)**. Το BindingResult είναι subinterface του interface Errors

insert.html

insert.html

```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org" lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Εγγραφή Χρήστη</title>
6  </head>
7  <body>
8      <div th:object="${studentInsertDto}">
9
10         <form action="insert" method="post">
11             <div>
12                 <span><b>Εγγραφή Χρήστη</b></span>
13             </div>
14             <div>
15                 <input id="firstname" type="text" placeholder="Όνομα" name="firstname" th:field="*{firstname}">
16             </div>
17             <div>
18                 <input id="lastname" type="text" placeholder="Επώνυμο" name="lastname" th:field="*{lastname}">
19             </div>
20             <input type="submit" value="Εισαγωγή">
21         </form>
22
23         <div th:if="${#fields.hasErrors('*')}">
24             <ul>
25                 <li th:each="err : ${#fields.errors('*')}" th:text="${err}" />
26             </ul>
27         </div>
28     </div>
29 </body>
30 </html>
```

- Στο Thymeleaf έχουμε το **studentInsertDto** που είναι συνδεδεμένο με τη φόρμα
- Όσο αφορά το Validation έχουμε το **#fields** object του Thymeleaf που θα δούμε στη συνέχεια

Data Binding Thymeleaf


```
23        <div th:if="${#fields.hasErrors('*')}">
24            <ul>
25                <li th:each="err : ${#fields.errors('*')}" th:text="${err}" />
26            </ul>
27        </div>
28    </div>
```

- Το #fields object είναι ένα Thymeleaf utility object που παρέχει πρόσβαση σε πληροφορίες για τα form fields και τα errors τους. Γίνεται populate από το BindingResult
- Η μέθοδος **#fields.hasErrors('*')** είναι μία utility μέθοδος που ελέγχει, αν υπάρχουν errors για κάθε ένα από τα πεδία της φόρμας. Αυτό σημαίνει το * wildcard. Δηλαδή να αναζητηθούν λάθη σε όλα τα πεδία της φόρμας
- Μπορούμε να παρουσιάσουμε όλα τα λάθη διασχίζοντας το object **${#fields.errors('*')}**

StudentInsertDTO


```java
package gr.aueb.cf.springstarter.dto;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

public class StudentInsertDTO {
    @NotBlank(message = "Not Blank")
    @Size(min = 3, max = 50)
    private String firstname;

    @NotBlank(message = "Not Blank")
    @Size(min = 3, max = 50)
    private String lastname;

    public String getFirstname() { return firstname; }
    public void setFirstname(String firstname) { this.firstname = firstname; }
    public String getLastname() { return lastname; }
    public void setLastname(String lastname) { this.lastname = lastname; }
}
```

- Τα annotations είναι μέρος του Bean validation

Student Validator (1)


```java
package gr.aueb.cf.springstarter.validator;

import gr.aueb.cf.springstarter.dto.StudentInsertDTO;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

@Component
public class StudentValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) { return StudentInsertDTO.class == clazz; }

}
```

- Ο ***StudentValidator*** κάνει implements τον Spring Validator. Η μέθοδος *supports* επιστρέφει true για κλάσεις τύπου StudentInsertDTO
- Χαρακτηρίζουμε ως **@Component** για να γίνει managed bean και να μπορεί να εισαχθεί με @Autowired

Student Validator (2)


```
17      @Override
18      public void validate(Object target, Errors errors) {
19          StudentInsertDTO studentInsertDto = (StudentInsertDTO) target;
20
21          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstname", "empty");
22          if (studentInsertDto.getFirstname().length() < 3 || studentInsertDto.getFirstname().length() > 50) {
23              errors.rejectValue("firstname", "size");
24          }
25
26          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastname", "empty");
27          if (studentInsertDto.getLastname().length() < 3 || studentInsertDto.getLastname().length() > 50) {
28              errors.rejectValue("lastname", "size");
29          }
30      }
31  }
```

- Με την κλάση **ValidationUtils** του Spring και τις static μεθόδους **rejectIfEmpty** και **rejectIfEmptyOrWhitespace** μπορούμε να ελέγξουμε για empty πεδία ή whitespaces. Παίρνει τρεις παραμέτρους: 1) το αντικείμενο ***errors***, στο οποίο γίνονται register τα τυχόν λάθη, 2) το ***πεδίο*** το οποίο ελέγχουμε (το αντικείμενο obj δεν χρειάζεται να περάσει ως παράμετρος γιατί το errors έχει εσωτερικά ένα reference στο αντικείμενο), και 3) το **error code**, το οποίο εμείς δίνουμε και χρησιμοποιείται από το *MessageSource* όπως θα δούμε στη συνέχεια για να εμφανίζουμε custom μηνύματα λάθους

Student Validator (3)


```
17      @Override
18      public void validate(Object target, Errors errors) {
19          StudentInsertDTO studentInsertDto = (StudentInsertDTO) target;
20
21          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstname", "empty");
22          if (studentInsertDto.getFirstname().length() < 3 || studentInsertDto.getFirstname().length() > 50) {
23              errors.rejectValue("firstname", "size");
24          }
25
26          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastname", "empty");
27          if (studentInsertDto.getLastname().length() < 3 || studentInsertDto.getLastname().length() > 50) {
28              errors.rejectValue("lastname", "size");
29          }
30      }
31  }
```

- Μπορούμε επίσης να κάνουμε custom ελέγχους για το length με τη χρήση της μεθόδου **errors.rejectValue** που παίρνει δύο παραμέτρους, *το όνομα του πεδίου* που κάνουμε validate και το *error code*
- Κάτι τέτοιο θα μπορούσε να γίνει και στο DTO bean με **@Size(min=3, max=64, message="invalid length")** για παράδειγμα

Validation

- Το *Bean Validation* και ο *Validator* μπορούν να χρησιμοποιούνται και ταυτόχρονα ώστε κάποια properties να τα ελέγχουμε με Bean Validation (π.χ. @Email) και κάποια άλλα properties (firstname, lastname, κλπ.) με Spring Validator

Insert Student validation (1)


```
48      @RequestMapping(path = "/students/insert", method = RequestMethod.GET)
49  @   public String getStudentForm(Model model) {
50          model.addAttribute("studentInsertDto", new StudentInsertDTO());
51          return "student/insert";
52      }
53
54      @RequestMapping(path = "/students/insert", method = RequestMethod.POST)
55      public String insertStudent(@Valid @ModelAttribute("studentInsertDto") StudentInsertDTO dto,
56                                  BindingResult bindingResult,Model model) {
57          studentValidator.validate(dto, bindingResult);
58          if (bindingResult.hasErrors()) {
59              return "student/insert";
60          }
61          model.addAttribute("dto", dto);
62          return "student/success";
63      }
```

- Στην insertStudent που χειρίζεται τα POST requests, έχουμε εισάγει το **@Valid** πριν το @ModelAttribute ώστε αν ελέγχει τα **bean validation** annotations

Με την κλήση της **μεθόδου validate** καλούμε τον Spring custom Validator (τον *StudentVaidator*)

Insert Student validation (2)


```
48      @RequestMapping(path = "/students/insert", method = RequestMethod.GET)
49  @   public String getStudentForm(Model model) {
50          model.addAttribute("studentInsertDto", new StudentInsertDTO());
51          return "student/insert";
52      }
53
54      @RequestMapping(path = "/students/insert", method = RequestMethod.POST)
55      public String insertStudent(@Valid @ModelAttribute("studentInsertDto") StudentInsertDTO dto,
56                                  BindingResult bindingResult,Model model) {
57          studentValidator.validate(dto, bindingResult);
58          if (bindingResult.hasErrors()) {
59              return "student/insert";
60          }
61          model.addAttribute("dto", dto);
62          return "student/success";
63      }
```

- Το object **bindingResult** κάνει register όλα τα τυχόν errors τα οποία στη συνέχεια ελέγχουμε με την ***hasErrors***
- Αν η hasErrors() επιστρέψει true τότε στέλνουμε πάλι στη σελίδα student/insert όπου τα errors κάνουν populate το #fields object του Thymeleaf

Localization μηνυμάτων

- Το τι μηνύματα θα εμφανίζουμε εξαρτάται από το αν έχουμε κάνει config σωστά το message source και να έχουμε ονομάσει σωστά τα errors στο αρχείο messages.properties
- Διαφορετικά δημιουργούνται λάθη πριν φτάσει το #fields να εμφανίσει σωστά τα λάθη

Message Source

- Το **MessageSource** είναι ένα **interface του Spring** που δίνει τη δυνατότητα οργάνωσης αλλά και localization των μηνυμάτων που θέλουμε να εμφανίζουμε στους χρήστες
- Η βασική ιδέα είναι ότι μπορούμε στην εφαρμογή μας να ορίσουμε **custom error codes και να συνδέσουμε στη συνέχεια τα error codes με custom μηνύματα**
- Όταν δημιουργείται ένα error με συγκεκριμένο error code, εμφανίζεται το μήνυμα που έχουμε ορίσει
- Τα μηνύματα τα συγκεντρώνουμε στο αρχείο **messages.properties** στο φάκελο **/src/main/resources**

Internationalization – Spring Boot

- Μπορούμε να ορίσουμε διάφορα **locales** και να χρησιμοποιείται εκείνο που ζητείται από τον requested HTTP Header. Τα locales τα ορίζουμε ονομάζοντας αρχεία **messages_xx.properties**, όπου **xx** το **language-code** (π.χ.. el-GR, en-US, κλπ.)
- **Αν θέλουμε να χρησιμοποιούμε πάντα ένα default locale** π.χ. Ελληνικά, τότε θα πρέπει να το ορίσουμε στο αρχείο *application.properties*
- Θα πρέπει να ορίσουμε και **το locale αλλά και τον localeResolver=fixed**. Αν δεν ορίσουμε *localeResolver*, χρησιμοποιείται ο *default resolver (AcceptHeaderLocaleResolver)* που εμφανίζει μηνύματα, ανάλογα με τον requested HTTP header)
- Αν δεν υπάρχουν *messages_xx.properties* αρχεία, γίνεται **fallback στο messages.properties**

```
10  spring.web.locale=el_GR
11  spring.web.locale-resolver=fixed
```

Validator & MessageCodes

- Στο αρχείο messages.properties εισάγουμε να βρει κατά σειρά τα παρακάτω (από το ειδικότερο στο γενικότερο):

1. code + "." + object name + "." + field
2. code + "." + field
3. code + "." + field type
4. code

MessageSourceAccessor

- Μπορούμε να έχουμε άμεση πρόσβαση σε μηνύματα του MessageSource μέσω του **MessageSourceAccessor** με την μέθοδο **getMessage(String code)** που επιστρέφει το αντίστοιχο μήνυμα σύμφωνα με το default locale
- Για παράδειγμα:

```
throw new BadCredentialsException(accessor.getMessage("badCredentials"));
```

StudentInsertDTO


```
7   public class StudentInsertDTO {
8       @Email
9       private String email;
10  //    @NotBlank(message = "Not Blank")
11  //    @Size(min = 3, max = 50)
12      private String firstname;
13
14  //    @NotBlank(message = "Not Blank")
15  //    @Size(min = 3, max = 50)
16      private String lastname;
17
18      public String getFirstname() { return firstname; }
21      public void setFirstname(String firstname) { this.firstname = firstname; }
24      public String getLastname() { return lastname; }
27      public void setLastname(String lastname) { this.lastname = lastname; }
30  }
```

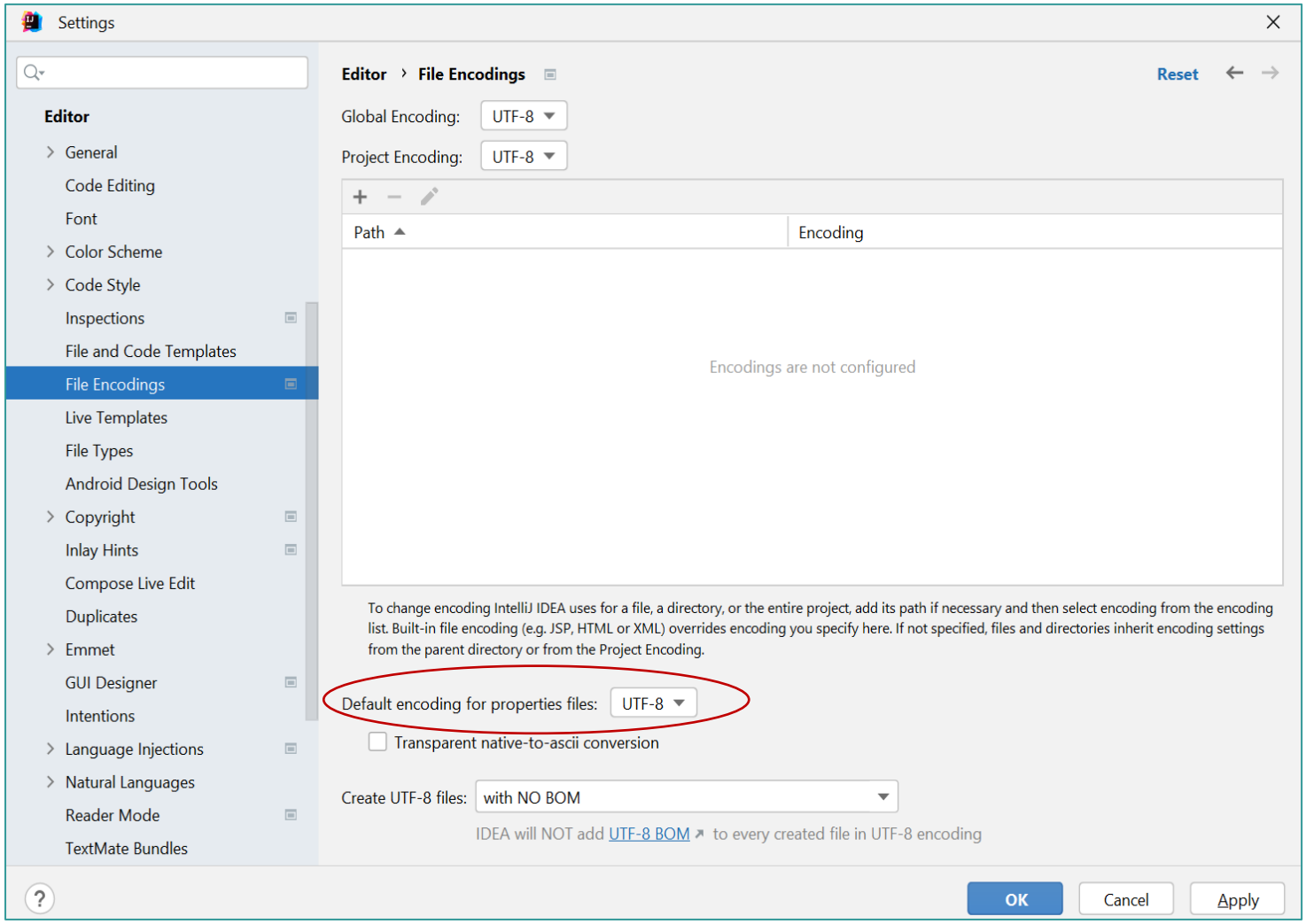
- Έστω ότι έχουμε ορίσει ένα νέο StudentInsertDTO έχοντας απενεργοποιήσει τα annotations στα firstname, lastname (comment out) και έχοντας προσθέσει ένα πεδίο email με annotation @Email

Validator


```java
@Component
public class StudentValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) { return StudentInsertDTO.class == clazz; }

    @Override
    public void validate(Object target, Errors errors) {
        StudentInsertDTO studentInsertDto = (StudentInsertDTO) target;

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstname", "empty");
        if (studentInsertDto.getFirstname().length() < 3 || studentInsertDto.getFirstname().length() > 50) {
            errors.rejectValue("firstname", "size");
        }

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastname", "empty");
        if (studentInsertDto.getLastname().length() < 3 || studentInsertDto.getLastname().length() > 50) {
            errors.rejectValue("lastname", "size");
        }
    }
}
```

- Τα firstname και lastname τα ελέγχουμε στον Validator

Properties files encoding



- File / Settings / Editor / File Encodings / Default Encoding for properties files: UTF-8
- Ώστε στο messages.properties να μπορούμε να γράφουμε Ελληνικά

Αρχείο messages.properties

messages.properties

```
1  empty.studentInsertDto.firstname=Συμπληρώστε το πεδίο Όνομα.
2  empty.studentInsertDto.lastname=Συμπληρώστε το πεδίο Επώνυμο.
3  size.studentInsertDto.firstname=Το πεδίο Όνομα πρέπει να έχει από 3 έως 30 χαρακτήρες.
4  size.studentInsertDto.lastname=Το πεδίο Επώνυμο πρέπει να έχει από 3 έως 30 χαρακτήρες.
5  Email=Εισάγετε ένα έγκυρο email
```

- Στο αρχείο **messages.properties** έχουμε τα μηνύματα της εφαρμογής μας

Αρχείο application.properties

application.properties

```
1  spring.web.locale=el_GR
2  spring.web.locale-resolver=fixed
3
```

- Όπως αναφέραμε, θα πρέπει να ορίσουμε και το **locale** (el_GR) αλλά και τον **localeResolver=fixed**
- Αν δεν ορίσουμε *localeResolver*, χρησιμοποιείται ο *default resolver (AcceptHeaderLocaleResolver)* που εμφανίζει μηνύματα, ανάλογα με τον requested HTTP header

Insert.html (1)


```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <title>Εγγραφή Χρήστη</title>
</head>
<body>
    <div th:object="${studentInsertDto}">

        <form action="insert" method="post">
            <div>
                <span><b>Εγγραφή Χρήστη</b></span>
            </div>
            <div>
                <input id="email" type="text" placeholder="E-mail" name="email" th:field="*{email}">
            </div>
            <div>
                <input id="firstname" type="text" placeholder="Όνομα" name="firstname" th:field="*{firstname}">
            </div>
            <div>
                <input id="lastname" type="text" placeholder="Επώνυμο" name="lastname" th:field="*{lastname}">
            </div>
            <input type="submit" value="Εισαγωγή">
        </form>
```

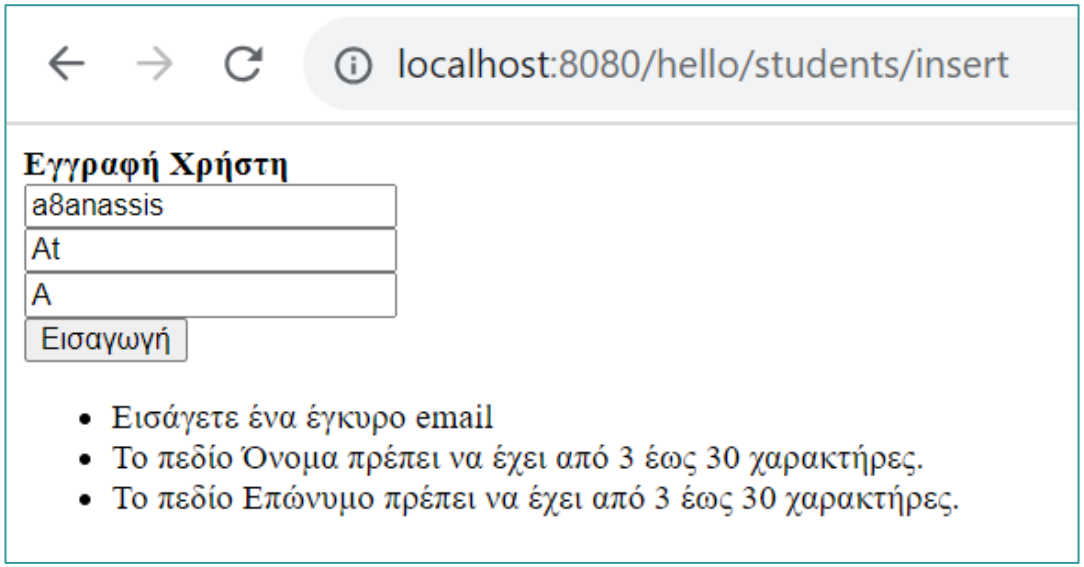
- Βλ. επόμενη διαφάνεια

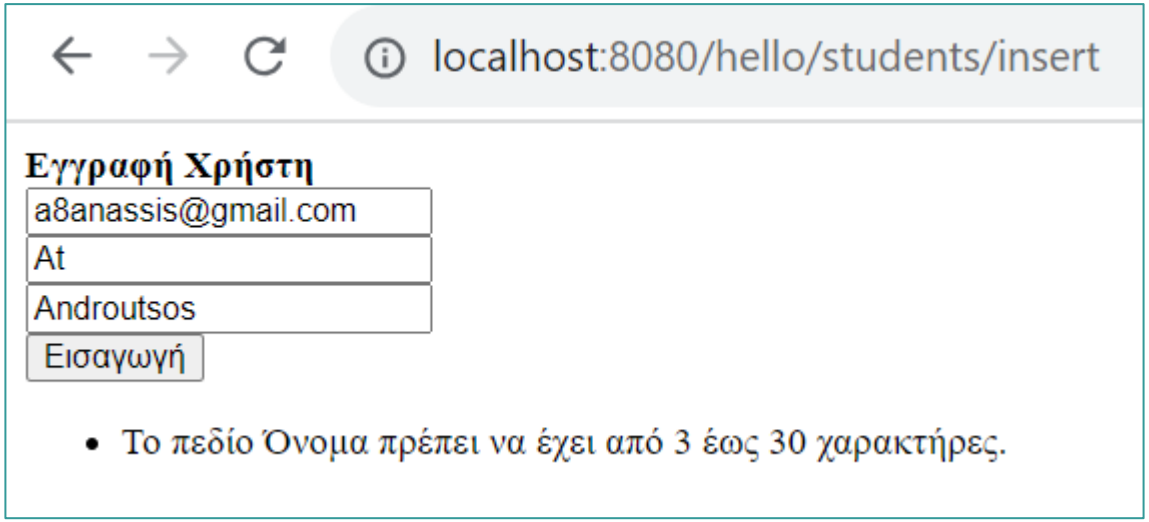
Insert.html (2)


```
25
26          <div th:if="${#fields.hasErrors('*')}">
27              <ul>
28                  <li th:each="err : ${#fields.errors('*')}" th:text="${err}" />
29              </ul>
30          </div>
31      </div>
32  </body>
33  </html>
```

- Error handling με #fields

Φόρμα Εισαγωγής





- Φόρμα εισαγωγής με Validation