HERMES
Back-End
System Architecture

Version: 1.1

Status: Released

Ιστορικό Αναθεωρήσεων

Date	Version	Description	Authors
<19/01/2021>	1.0	Initial version	Nick Dimitrakopoulos Paraskevi-Theofania Gourgioti Ioannis Christou
<18/02/2021>	1.1	API Reference added	Nick Dimitrakopoulos Paraskevi-Theofania Gourgioti Ioannis Christou

Πίνακας Περιεχομένων

Εισαγωγή

Σκοπός αυτού του εγγράφου

Σκοπός αυτού του εγγράφου είναι να τεκμηριωθεί η αρχιτεκτονική του Back-End του Πληροφοριακού Συστήματος (εφεξής ΠΣ) ΕΡΜΗΣ. Απευθύνεται σε μηχανικούς λογισμικού που επιθυμούν να κατανοήσουν, να συντηρήσουν ή να επεκτείνουν το συγκεκριμένο υποσύστημα.

Ορισμός, Ακρωνύμια και Συντομογραφίες

Όρος	Σημασία
Keycloak	Λογισμικό ανοιχτού κώδικα, το οποίο παρέχει Single-Sign-On (SSO) και Identity and Access Management (IAM) δυνατότητες σε μοντέρνες εφαρμογές και υπηρεσίες.

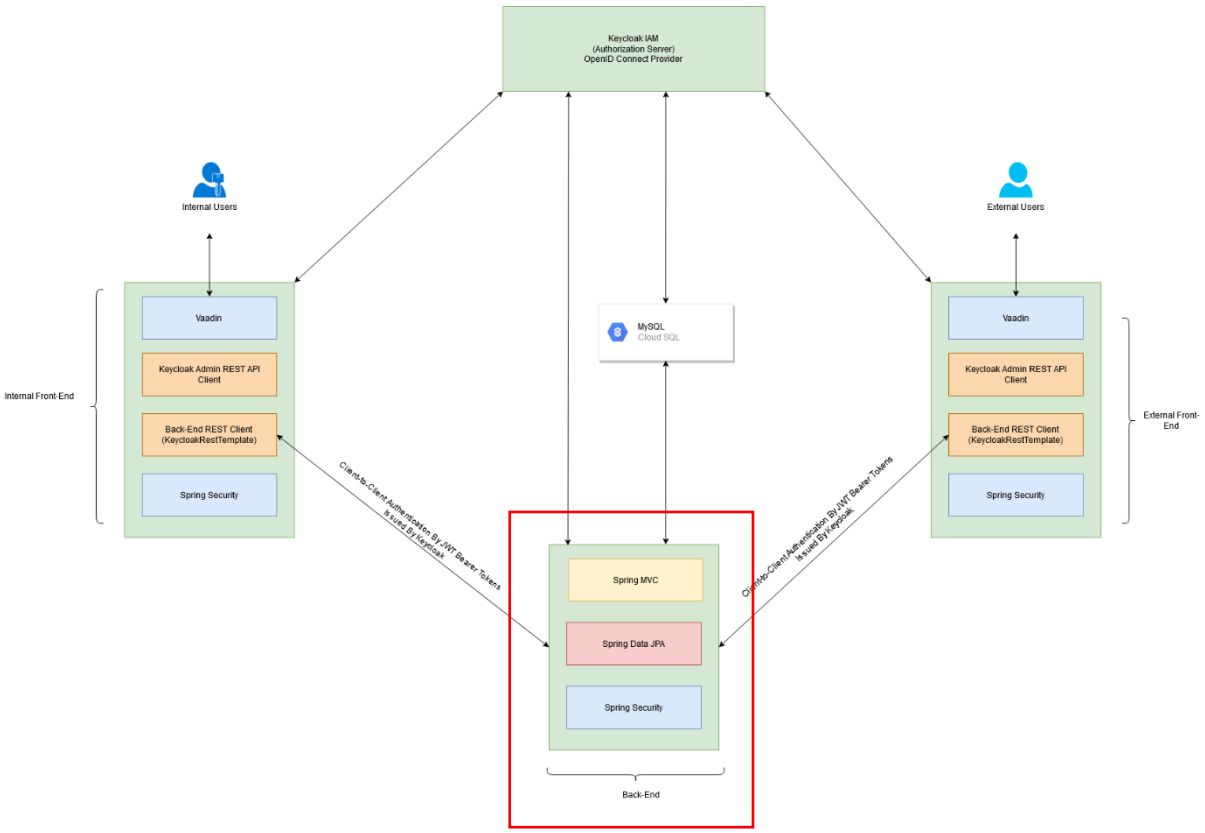
Λίγα λόγια για το Back-End υποσύστημα

Το Back-End υποσύστημα έχει ως στόχο την διαχείριση των αιτήσεων κάθε οργανισμού που είναι ενταγμένος στο ΠΣ ΕΡΜΗΣ. Παρέχει ένα REST API, το οποίο **δεν** είναι εκτεθειμένο στο δημόσιο ίντερνετ και μπορεί να χρησιμοποιηθεί από οποιοδήποτε άλλο υποσύστημα που έχει ως απαίτηση την παροχή υπηρεσιών management των αιτήσεων.

Κύριοι στόχοι του υποσυστήματος

- Παροχή management υπηρεσιών για τις αιτήσεις στους υπαλλήλους του κάθε οργανισμού.
- Παροχή management υπηρεσιών για τις αιτήσεις στους πολίτες.
- Να παρέχει τις παραπάνω υπηρεσίες με ασφάλεια μέσω του IAM Keycloak, χρησιμοποιώντας Access Control μηχανισμούς όπως:
    - Attribute-based access control (ABAC)
    - Role-based access control (RBAC)
    - User-based access control (UBAC)

Το Back-End ως μέρος του συνολικού ΠΣ



Εξαρτήσεις

Για να θεωρείται διαθέσιμο το Back-End ως υπηρεσία πρέπει να είναι διαθέσιμα και λειτουργικά:

- Ένα instance της **MySQL**
- Ένα instance του **Keycloak**

Εξαρτώμενες υπηρεσίες

Το Back-End παρέχει υπηρεσίες στα ακόλουθα υποσυστήματα του ΠΣ ΕΡΜΗΣ:

- Στο **Internal Front-End**, το οποίο απευθύνεται στους εσωτερικούς χρήστες του ΠΣ.
- Στο **External Front-End**, το οποίο απευθύνεται στους εξωτερικούς χρήστες του ΠΣ.

Αρχιτεκτονική

Dependencies

Το Back-End χρησιμοποιεί το **Spring Boot 2.3.8** ως κύριο framework. Τα κύρια dependencies του είναι τα:

- **Spring Data JPA**
- **Spring Web**
- **Spring Security**
- **Keycloak Client Adapter For Spring Boot (12.0.2)**

Data Model

AbstractEntity

Κάθε entity που υπάρχει ήδη ή πρόκειται να δημιουργηθεί στο μέλλον πρέπει να κάνει extend την abstract κλάση **AbstractEntity**. Η **AbstractEntity** φροντίζει να παρέχει σε κάθε οντότητα τα βασικά πεδία που απαιτούνται για:

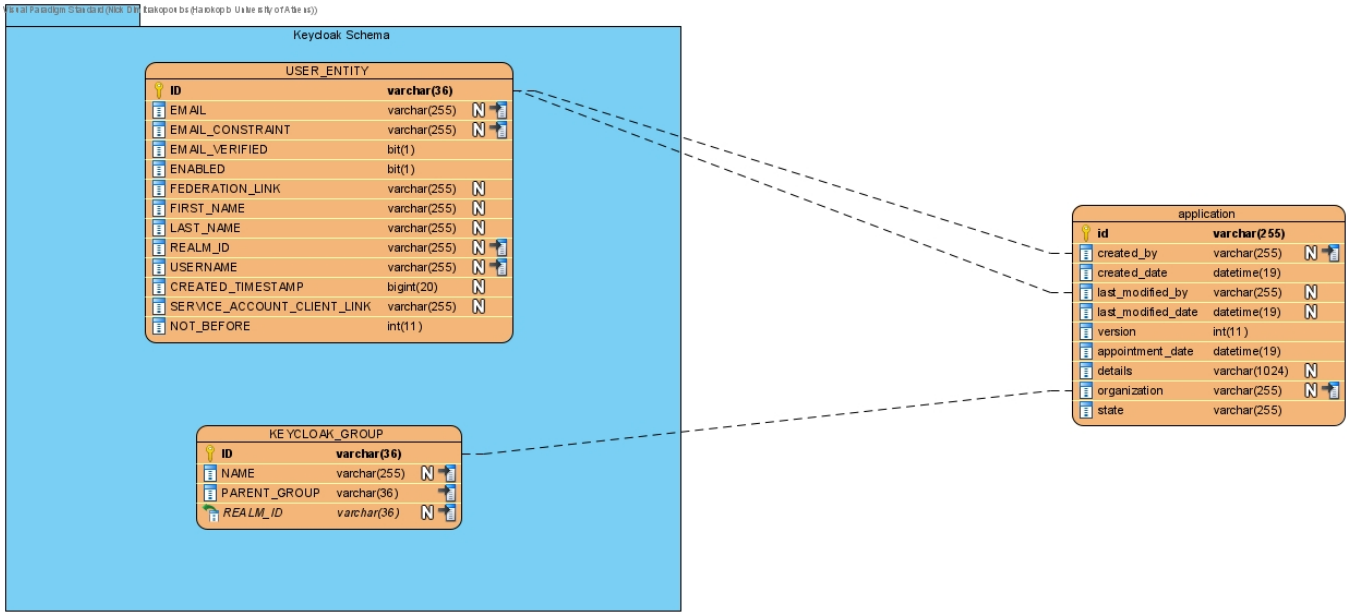
- **Την ταυτοποίηση της οντότητας**: PK τύπου String σε UUID (Universally Unique Identifier) μορφή
- **Βασικό Auditing για κάθε οντότητα** (ποιος/α και πότε δημιούργησε ή τροποποίησε τελευταία φορά την οντότητα. Για περισσότερες πληροφορίες ανατρέξτε εδώ.
- **Υποστήριξη locking για αποτροπή ανωμαλιών κατά τη ταυτόχρονη πρόσβαση των δεδομένων.** Χρησιμοποιείται Optimistic Locking, για περισσότερες πληροφορίες ανατρέξτε εδώ.

Application

Η οντότητα των αιτήσεων των πολιτών. Κληρονομεί όλα τα χαρακτηριστικά του AbstractEntity και επιπρόσθετα έχει τα εξής γνωρίσματα:

- **ID πολίτη** που κάνει την αίτηση (χρησιμοποιούμε το **created_by** του auditing γιατί μόνο ο πολίτης έχει δικαίωμα δημιουργίας αίτησης)
- **ID οργανισμού** στον οποίο καταθέτεται η αίτηση
- **Κατάσταση της αίτησης** (περιγράφεται στο Functional Requirements έγγραφο)
- **Λεπτομέρειες σχετικά με την αίτηση** (σχόλια)
- **Ημερομηνία και ώρα που επιθυμεί ο πολίτης να γίνει η συνάντηση**

Τα ID πολίτη και ID οργανισμού προέρχονται από το Keycloak. Η πληροφορία μεταδίδεται μέσω του token.



Εικόνα 1 Η έμμεση συσχέτιση του σχήματος του Keycloak με αυτό του Back-End

Data Access Layer

Ως Data Access Layer χρησιμοποιείται το **Spring Data JPA**. Μέσω των **JpaRepository<T,ID>** interfaces μπορούμε να έχουμε τις βασικές μεθόδους για να κάνουμε CRUD operations χωρίς να γράφουμε boilerplate κώδικα.

Η ενσωμάτωση των access control μηχανισμών γίνεται με τη χρήση του **@Query** . Μπορούμε να χρησιμοποιήσουμε **SpEL (Spring Expression Language)** ώστε να παράγουμε queries τα οποία χρησιμοποιούν πληροφορίες από το Access Token του Keycloak.

Για παράδειγμα:

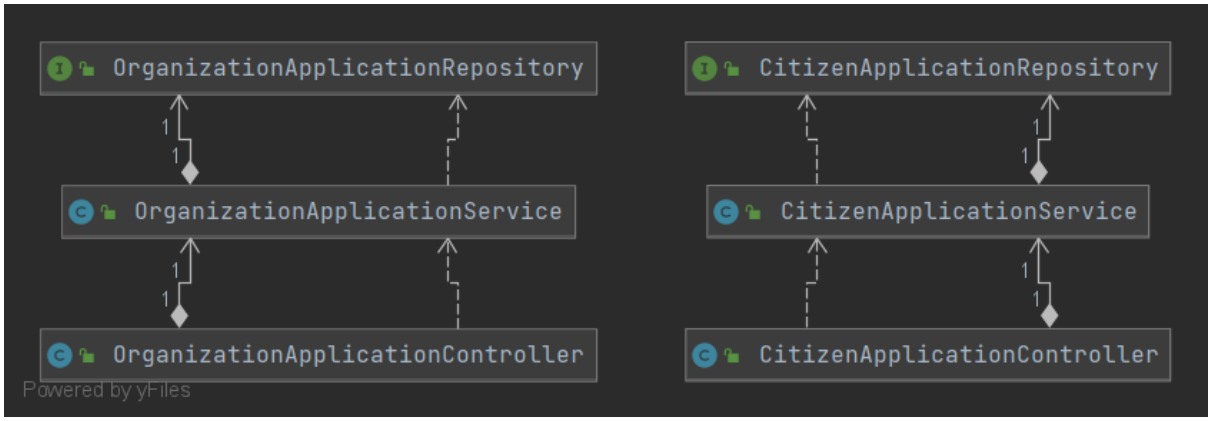
```
@Query("SELECT a FROM Application a WHERE a.createdBy =
?#{authentication.name}")
Page<Application> findAll(Pageable pageable);
```

Αυτό το query φιλτράρει τη σελίδα των αιτήσεων που θα επιστραφεί βάσει το username του χρήστη που έκανε το request. Οποιαδήποτε πληροφορία υπάρχει στο Token μπορεί να χρησιμοποιηθεί στο @Query.

Μπορείτε να ανατρέξετε εδώ για περισσότερες λεπτομέρειες σχετικά με αυτή τη δυνατότητα.

REST Endpoints

Για την κατασκευή των endpoints έχει χρησιμοποιηθεί η κλασσική προσέγγιση του γνωστού Repository-Service-Controller pattern.



To Repository-Service-Controller Pattern

API Reference

URL	/organization/application
Method	GET
Path parameters	-
Query parameters	• offset (int) -> Παρέλειψε τις πρώτες offset αιτήσεις • limit (int) -> Όριο επιστροφής αιτήσεων
Body	-
Authorization	Bearer Token
Example Response (Success)	**application/json** [{ "id": "42f25749-f615-49c9-8122-efcfc1684f43", "version": 2, "created_date": "2021-01-30T21:09:03.000+00:00", "last_modified_date": "2021-01-30T21:13:31.000+00:00", "created_by": "delta",

	"last_modified_by": "delta",
	"organization": "6636bf6a-a259-4601-95f9-3ae657182844",
	"state": "SUBMITTED",
	"details": "",
	"appointment_date": "2021-01-31T01:30:00.000+00:00"
	},
	{
	"id": "494f910c-1904-442c-9036-c7ad10335051",
	"version": 1,
	"created_date": "2021-01-30T21:08:13.000+00:00",
	"last_modified_date": "2021-01-30T21:08:34.000+00:00",
	"created_by": "delta",
	"last_modified_by": "delta",
	"organization": "6636bf6a-a259-4601-95f9-3ae657182844",
	"state": "CANCELED",
	"details": "Please see my request",
	"appointment_date": "2021-01-31T01:30:00.000+00:00"
	}
	]

Response status code	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal server error

URL	/organization/application/{id}
Method	GET
Path parameters	id (string) -> το μοναδικό αναγνωριστικό της αίτησης
Query parameters	-
Body	-
Authorization	Bearer Token
Example Response	**application/json**

	{
	"id": "42f25749-f615-49c9-8122-efcfc1684f43",
	"version": 2,
	"created_date": "2021-01-30T21:09:03.000+00:00",
	"last_modified_date": "2021-01-30T21:13:31.000+00:00",
	"created_by": "delta",
	"last_modified_by": "delta",
	"organization": "6636bf6a-a259-4601-95f9-3ae657182844",
	"state": "SUBMITTED",
	"details": "Test",
	"appointment_date": "2021-01-31T01:30:00.000+00:00"
	}

Response status code	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal server error

URL	/organization/application/{id}
Method	PUT
Path parameters	id (string) -> το μοναδικό αναγνωριστικό της αίτησης
Query parameters	-

Body	**application/json**
	
	{
	"organization": "auth",
	"state": "COMPLETED",
	"details": "I want nice service",
	"appointment_date": "30-12-2020T17:32:24"
	}

Authorization	Bearer Token
Example Response	1
Response status code	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal server error

Exception Handling

Κατά τη διάρκεια της λειτουργίας του Back-End είναι παραπάνω από πιθανό να συμβούν κάποιες αναμενόμενες (ή μη) εξαιρέσεις. Κοινοί τύποι εξαιρέσεων αποτελούν:

- Η μη εύρεση ενός ζητούμενου πόρου (404)
- Αποτυχία στο validation μιας δοσμένης οντότητας (Hibernate Validator)
- Μια συγκρουόμενη ενημέρωση μιας οντότητας (Optimistic Locking)
- Κάποιο απροσδόκητο Internal Server Error (500)

Το Spring Web προσφέρει εξελιγμένο μηχανισμό διαχείρισης σφαλμάτων, ο οποίος μας βοηθάει στο να αποφεύγουμε να γράφουμε boilerplate και unreadable try-catch κώδικα. Οι **@ExceptionHandler** ενεργοποιούνται όποτε πετάγεται η συγκεκριμένη εξαίρεση που έχει δηλωθεί στο value του

annotation. Η μέθοδος στην οποία γίνεται annotate ο **@ExceptionHandler** πλέον καθορίζει τι θα επιστραφεί ως απάντηση στο request.

```
//Exception controller when @Valid fails
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<?>
handleValidationException(MethodArgumentNotValidException ex)
{
    List<String> errors = ex.getBindingResult().getAllErrors()
            .stream()
            .map(x-> x.getDefaultMessage())
            .collect(Collectors.toList());

    return ResponseEntity.badRequest().body(Map.of("errors",errors));
}
```

Ένα παράδειγμα εφαρμογής του @ExceptionHandler