HERMES
Front-Ends
System Architecture

Version: 1.0

Status: Released

Ιστορικό Αναθεωρήσεων

Date	Version	Description	Authors
<14/02/2021>	1.0	Initial version	Nick Dimitrakopoulos Paraskevi-Theofania Gourgioti Ioannis Christou

Πίνακας Περιεχομένων

Εισαγωγή

Σκοπός αυτού του εγγράφου

Σκοπός αυτού του εγγράφου είναι να τεκμηριωθεί η αρχιτεκτονική των Front-End του Πληροφοριακού Συστήματος (εφεξής ΠΣ) ΕΡΜΗΣ. Απευθύνεται σε μηχανικούς λογισμικού που επιθυμούν να κατανοήσουν, να συντηρήσουν ή να επεκτείνουν τα συγκεκριμένα υποσυστήματα.

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.
Vaadin	Πλατφόρμα ανοιχτού κώδικα, η οποία παρέχει δυνατότητα κατασκευής ποιοτικών Web User Interfaces (Front-End), χρησιμοποιώντας Java.

Λίγα λόγια για τα Front-End υποσυστήματα

Το ΠΣ Ερμής έχει 2 front-end υπηρεσίες:

- Το **Internal Front-End**, το οποίο αναλαμβάνει να προσφέρει UI στους εσωτερικούς χρήστες του συστήματος, αξιοποιώντας πόρους από το REST API του Keycloak καθώς από το REST API του back-end υποσυστήματος.
- Το **External Front-End**, το οποίο αναλαμβάνει να προσφέρει UI στους εξωτερικούς χρήστες του συστήματος, αξιοποιώντας επίσης πόρους από το REST API του Keycloak καθώς από το REST API του back-end υποσυστήματος.

Και τα 2 Front-End χρησιμοποιούν το Vaadin Flow ως τεχνολογία υλοποίησης.

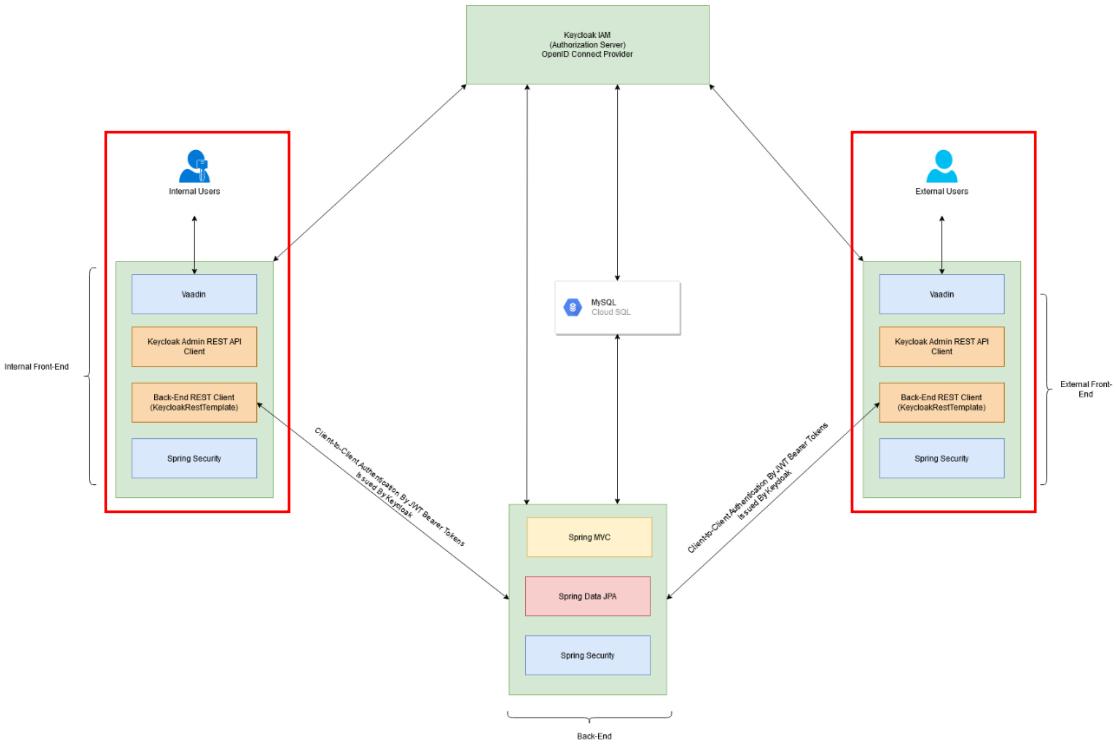
Κύριοι στόχοι του Internal Front-End υποσυστήματος

- Παροχή GUI στους/στις **διαχειριστές/στριες οργανισμών** για τη **διαχείριση οργανισμών**, **προϊσταμένων οργανισμών** και **πολιτών.**
- Παροχή GUI στους/στις **προϊσταμένους/ες οργανισμών** για τη **διαχείριση των υπαλλήλων** των οργανισμών.
- Παροχή GUI στους/στις **υπαλλήλους οργανισμών** για τη **διαχείριση των αιτήσεων** των οργανισμών.
- Να παρέχει τις παραπάνω υπηρεσίες **με ασφάλεια** μέσω του IAM Keycloak.

Κύριοι στόχοι του External Front-End υποσυστήματος

- Παροχή GUI στους πολίτες ώστε, να διαχειρίζονται τα ραντεβού τους με τους διάφορους οργανισμούς του συστήματος.
- Να παρέχει τις παραπάνω υπηρεσίες **με ασφάλεια** μέσω του IAM Keycloak.

Τα Front-Ends ως μέρος του συνολικού ΠΣ



Εξαρτήσεις

Για να θεωρείται διαθέσιμο τα Front-Ends ως υπηρεσία πρέπει να είναι διαθέσιμα και απολύτως λειτουργικά:

- Instance του **Keycloak** για **authentication/authorization.**

Από εκεί και πέρα ανάλογα την περίπτωση, τα ακόλουθα services πρέπει να είναι διαθέσιμα, αλλά ενδεχόμενο downtime δεν επηρεάζει ολοκληρωτικά την λειτουργία του front-end παρά μόνο σε τομείς :

- Instance του **Back-End** για τη διαχείριση των ραντεβού.
- Instance του **Keycloak** με τα Citizen & Organization endpoints απολύτως λειτουργικά.

Αρχιτεκτονική

Frameworks - Libraries

Τα Front-Ends χρησιμοποιούν **Spring Boot 2.3.8 + Vaadin Flow 14.4.7 LTS**. Άλλα κύρια modules είναι τα:

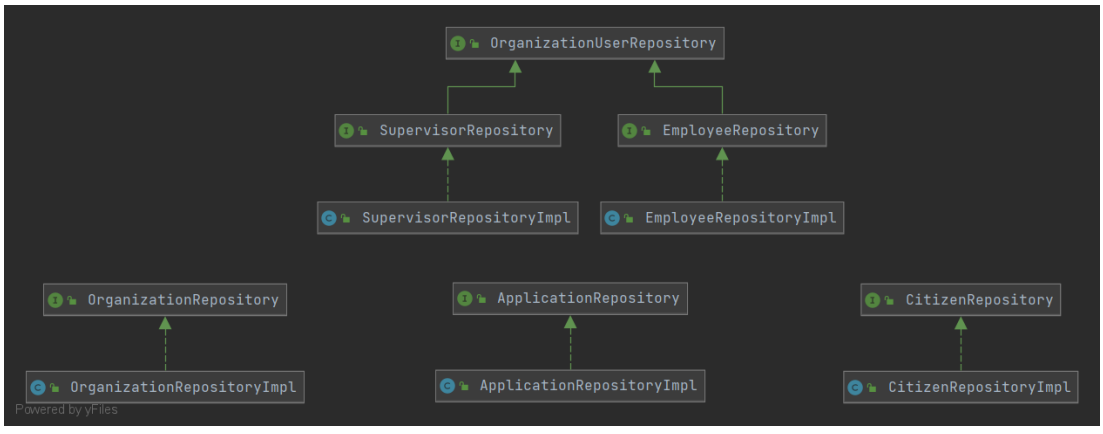
- **Spring Web**
- **Spring Security**
- **Vaadin Security Spring Boot Starter 1.0.0 (by Patrick Schmidt)**
- **Keycloak Client Adapter For Spring Boot (12.0.2)**

Τα Layers της εφαρμογής

Για την εφαρμογή επιλέχθηκε μία Abstraction (Layering) αρχιτεκτονική. Κάθε στρώμα επιτελεί ένα συγκεκριμένο έργο και αναλαμβάνει ένα «κομμάτι» της συνολικής λογικής. Παρακάτω, τα στρώματα αυτά, περιγράφονται αναλυτικά.

Repository Layer

Στο Repository Layer κατασκευάζουμε μέσω interfaces τις μεθόδους για την επικοινωνία με τα άλλα υποσυστήματα. Τα interfaces αυτά ακολουθούνται από implementations, στα οποία χρησιμοποιείται σε καθένα από αυτά ο κατάλληλος client (Keycloak Admin Client για επικοινωνία με το Keycloak ή KeycloakRestTemplate για επικοινωνία με το back-end). Τα interfaces καλύπτουν συνήθως μόνο κλασσική CRUD λειτουργικότητα.



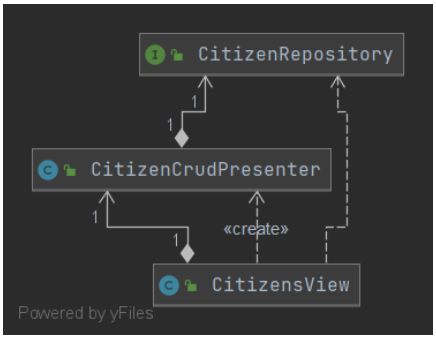
UML διάγραμμα των Repositories (Εσωτερικό Σύστημα)

Presenter Layer

Στο Presenter Layer ουσιαστικά «συνδέουμε» τα Views με τα Repositories (σαν facade) για να διαχειριστούμε κατάλληλα ενδεχόμενα exceptions που μπορεί να προκύπτουν κατά τη διάρκεια της επικοινωνίας. Προσθέτουμε ένα επιπλέον στρώμα ώστε να αποφύγουμε να έχουμε exception-handling κώδικα στα Views.

View Layer

Στο View Layer ουσιαστικά κατασκευάζεται το Web Component που τελικά προβάλλεται στον χρήστη. Ορίζουμε το UI και συνδέουμε τους Presenters για να έχουμε πρόσβαση στους πόρους.



UML διάγραμμα που δείχνει το Repository-Presenter-View Pattern στην περίπτωση των Citizens

Data Binding

Όλα τα views της εφαρμογής χρησιμοποιούν το CRUD Component. Το component αυτό χρησιμοποιεί ένα Grid Component. Το Grid προσφέρει μέσω των DataProviders έναν εύκολο τρόπο για να κάνεις bind στο Grid δεδομένα από Back-End services.

Προτιμήσαμε να χρησιμοποιήσουμε τον lazy-loading τρόπο ώστε να μπορεί η υπηρεσία μας να αντέχει μεγάλο αριθμό οντοτήτων. Για να γίνει εφικτό αυτό, όλα τα services μας έπρεπε να υποστηρίζουν pagination (δηλαδή να έχουν ένα endpoint που να επιστρέφει τον συνολικό αριθμό οντοτήτων και άλλο ένα που επιστρέφει σελιδοποιημένα βάσει offset και limit τις οντότητες).

Η συμπεριφορά του lazy DataProvider είναι η εξής: Πρώτα καλεί το count endpoint. Μετά υπολογίζει μόνο του τις μεταβλητές offset και limit και καλεί το fetch endpoint ώστε να λάβει τις οντότητες. Ο DataProvider ενεργοποιείται αυτόματα ανάλογα τη συμπεριφορά του χρήστη, δηλαδή αν πχ επιθυμεί να κάνει scroll πιο κάτω, τότε ο DataProvider προνοεί και κάνει fetch το επόμενο σετ οντοτήτων πριν φανεί στον χρήστη η καθυστέρηση για τη λήψη τους.

Στους DataProviders «δένουμε» τους Presenters μας και ουσιαστικά έχουμε και exception handling, το οποίο περιγράφεται σε παρακάτω ενότητα.

```
this.getGrid().setDataProvider(DataProvider.fromCallbacks(
        fetch ->
presenter.findAll(fetch.getOffset(),fetch.getLimit()).stream(),
        count -> presenter.count()));
```

Παράδειγμα DataProvider

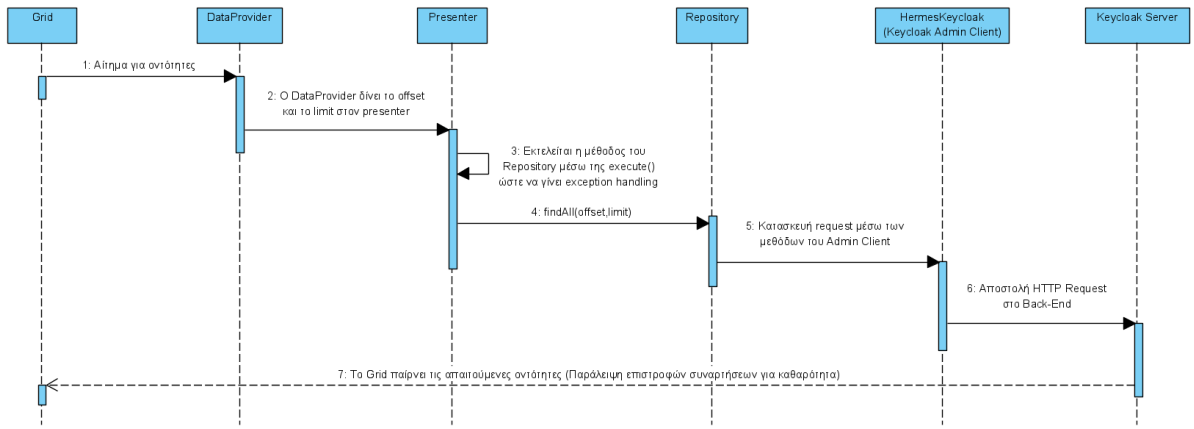
Επικοινωνία με εξωτερικά Services

Επικοινωνία με το Keycloak

Για να επικοινωνούν τα front-ends με το REST API του Keycloak, χρησιμοποιείται μια custom παραλλαγή του Keycloak Admin REST Client. Μέσω RESTEasy προστέθηκαν τα custom endpoints που έχουν ενσωματωθεί στην εικόνα του Keycloak. Σε ένα πιο production σενάριο θα έπρεπε να δημιουργηθεί ξεχωριστό artifact για την συντήρηση της βιβλιοθήκης, αλλά εδώ απλά σε κάθε front-end project βάζουμε ένα αντίγραφο του source της custom βιβλιοθήκης του client.

```
private final HermesKeycloak client;

@Value("${keycloak.realm}")
private String realm;

public CitizenRepositoryImpl(HermesKeycloak client) {
    this.client = client;
}


@Override
public Optional<UserRepresentation> findById(String id) {
    return
Optional.ofNullable(client.citizens().citizen(id).toRepresentation());
}
```

Παράδειγμα χρήσης HermesKeycloak



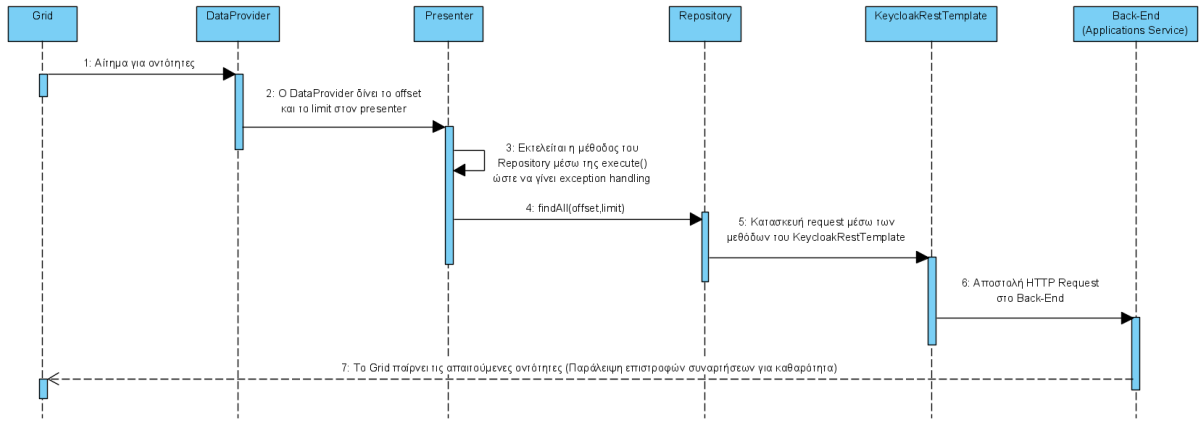
Sequence διάγραμμα που δείχνει τη ροή επικοινωνίας μεταξύ των Front-End και του Keycloak. Στο διάγραμμα αυτό προϋποθέτουμε για απλότητα ότι ο DataProvider ήδη ξέρει το αποτέλεσμα της count.

Επικοινωνία με το Back-End (Applications Service)

Για να επικοινωνούν front-ends με το REST API του Keycloak, χρησιμοποιείται το KeycloakRestTemplate, Το KeycloakRestTemplate είναι ένα extension του RestTemplate του Spring και χρησιμοποιείται για να αυτοματοποιήσει την παροχή των κατάλληλων έξτρα πληροφοριών (πχ authentication header).

```
private final KeycloakRestTemplate client;

@Value("${hermes.backend.url}")
private String baseURL;

public ApplicationRepositoryImpl(KeycloakRestTemplate client){
    this.client = client;
    this.client.setErrorHandler(new RestTemplateResponseErrorHandler());
}

@Override
@SneakyThrows(URISyntaxException.class)
public List<Application> findAll(int offset, int limit) {
    var url = new URIBuilder(baseURL)
            .setPathSegments("organization","application")
            .addParameter("offset",String.valueOf(offset))
            .addParameter("limit",String.valueOf(limit))
            .build()
            .toString();

    return client.exchange(url,HttpMethod.GET,null,
            new
ParameterizedTypeReference<List<Application>>(){}).getBody();
}
```

Παράδειγμα χρήσης KeycloakRestTemplate



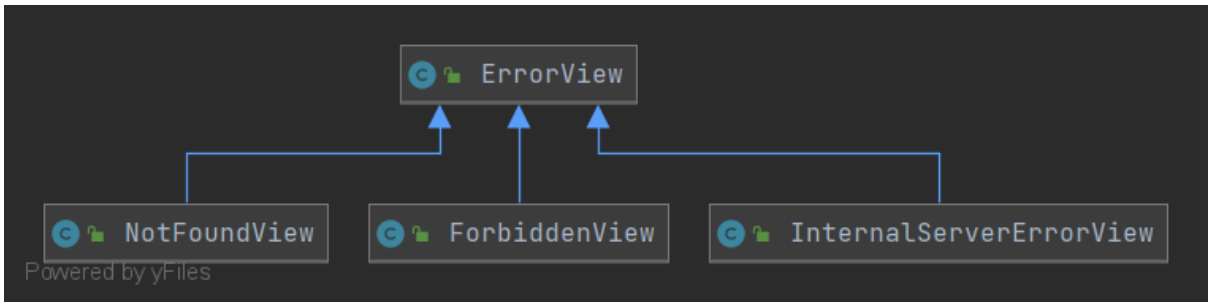
Sequence διάγραμμα που δείχνει τη ροή επικοινωνίας μεταξύ των Front-End και του Back-End. Στο διάγραμμα αυτό προϋποθέτουμε για απλότητα ότι ο DataProvider ήδη ξέρει το αποτέλεσμα της count

Exception Handling

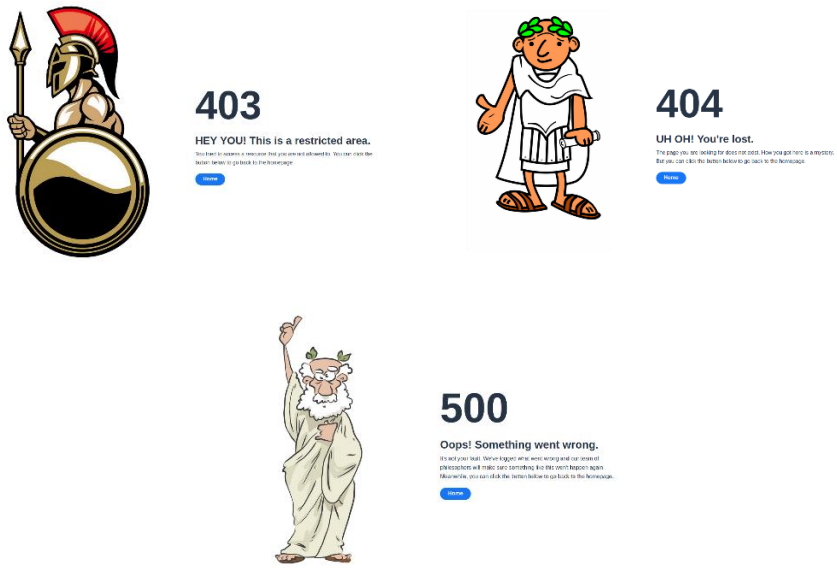
Υπάρχουν 2 ειδών εξαιρέσεων:

- Εξαιρέσεις κατά την πλοήγηση (navigation)
- Εξαιρέσεις κατά το Runtime (οποιαδήποτε άλλη uncaught εξαίρεση εκτός της φάσης του navigation).

Για τις εξαιρέσεις κατά την πλοήγηση, το Vaadin διαχερίζεται με όμορφο και decoupled τρόπο το τι πρέπει να συμβεί ανά τύπο εξαίρεσης. Συγκεκριμένα, παρέχει μία κλάση HasErrorParameter<T extends Exception> η οποία έχει συμπεριφορά παρόμοια με αυτή του @ExceptionHandler του Spring Boot και «πυροδοτείται» κάθε φορά που συμβαίνει η εξαίρεση που είναι δηλωμένη στο generic. Επιλέξαμε να διαχειριζόμαστε τις εξαιρέσεις αναγάγοντάς το σε γενικά error statuses που συναντάμε στο HTTP πρωτόκολλο (403,404,500). Για περισσότερες πληροφορίες μπορείτε να ανατρέξετε σε αυτόν τον σύνδεσμο.



Η ιεραρχία των error views που είναι δημιουργημένα με Polymer Templates



Διάφορα views που εμφανίζονται κατά το exception handling

Η λογική του navigation υλοποιείται σε κάθε view στην μέθοδο setParameter(BeforeEvent, String).

```
@Override
public void setParameter(BeforeEvent event, @OptionalParameter String
orgName)
{
    if (orgName != null) {
        var organization = getEditor().getItem();
        if (organization != null && orgName.equals(organization.getName()))
{
            return;
        }
        try {
            organization = organizationsPresenter
                    .findById(orgName)
                    .orElseThrow(NotFoundException::new);
            edit(organization, EditMode.EXISTING_ITEM);
        } catch (Exception ex){
            throw new RuntimeException(ex);
        }

    } else {
        setOpened(false);
    }
}
```

Παράδειγμα exception handling κατά το navigation για το view OrganizationsView. Ο presenter φροντίζει να αλληλεπιδρά με το view και να πετάει τις κατάλληλες εξαιρέσεις όταν είναι αναγκαίο ώστε να γίνουν handled.

Όσο αφορά τις εξαιρέσεις κατά το runtime, δυστυχώς δεν υπάρχει κάποιος «όμορφος» τρόπος να γίνει το exception handling. Προσπαθήσαμε μέχρι να επικοινωνήσουμε και με τους devs του Vaadin για να σχεδιάσουμε κάποιο αποδοτικό design pattern, αλλά καταλήξαμε στην απόφαση ότι ένα τέτοιο σενάριο δεν ταιριάζει στην φιλοσοφία του framework (ακόμα τουλάχιστον). Οπότε, το handling αυτών των εξαιρέσεων το αναλαμβάνει πάλι ο presenter.

```
protected <V> V execute(Callable<V> callable) throws Exception
{
    if(view == null) throw new IllegalStateException("View has not been
set");
    try{
        return callable.call();
    } catch (ConflictException ex){
        //In case of conflict inform the user about it.
        //Unfortunately, keycloak's API doesn't return any info on what
caused the conflict.
        view.showNotification("A conflict has occurred. " +
ex.getMessage());
        throw ex;
    } catch (NotFoundException ex) {
        throw ex; //We don't want to show a notification in this case.
    } catch (Exception ex){
        log.error(ex);
        view.showNotification("Something went wrong. Please try executing
the same action again.");
        throw ex;
    }
}
```

Exception Handling στον OrganizationCrudPresenter

Views

Σε αυτήν την ενότητα, περιγράφεται περιληπτικά κάθε διαθέσιμο view που υπάρχει στα 2 Front-End υποσυστήματα.

Εσωτερικό Σύστημα

Name:	HomeView
Roles:	Δεν έχει Role Access
Short Description:	Αρχική οθόνη που βλέπει κάθε τύπος χρήστη..
Base Path:	• / • /home (ισοδύναμο)
Path Parameters:	-
Data Sources:	-
Exceptions:	-

Name:	OrganizationsView
Roles:	ROLE_ORGS_ADMIN
Short Description:	Διαχείριση οργανισμών από τους διαχειριστές οργανισμών.
Base Path:	/organizations/{organization}
Path Parameters:	• organization (string) → Το μοναδικό αναγνωριστικό του οργανισμού Αν δοθεί {organization}, το οποίο αντιστοιχεί σε οργανισμό, ανοίγει η edit φόρμα. Αν δεν βρεθεί ο ορργανισμός, τότε επιστρέφεται 404.
Data Sources:	• Keycloak -> για τους οργανισμούς
Exceptions:	• **ConflictException** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **NotFoundException** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

Name:	CitizensView
Roles:	ROLE_ORGS_ADMIN
Short Description:	Διαχείριση πολιτών από τους διαχειριστές οργανισμών.
Base Path:	/citizens/{id}
Path Parameters:	• id (string) → Το μοναδικό αναγνωριστικό του πολίτη Αν δοθεί id, το οποίο αντιστοιχεί σε πολίτη, ανοίγει η edit φόρμα. Αν δεν βρεθεί το id, τότε επιστρέφεται 404
Data Sources:	• Keycloak -> για τους οργανισμούς
Exceptions:	• **ConflictException** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **NotFoundException** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

Name:	SupervisorsView
Roles:	ROLE_ORG_SUPERVISOR
Short Description:	Διαχείριση υπάλληλών του οργανισμού στον οποίο εργάζεται.
Base Path:	/organizations/supervisors/{organization}/{id}
Path Parameters:	• organization (String) → το όνομα του οργανισμού του supervisor Σε περίπτωση που το όνομα του οργανισμού δεν υπάρχει -> 404 Σε περίπτωση γενικού σφάλματος με το Keycloak instance -> 500 • id (string) → το id του υπάλληλου Αν δεν υπάρχει ο χρήστης -> 404 Σε περίπτωση γενικού σφάλματος με το Keycloak instance -> 500 **ΠΡΟΣΟΧΗ:** Σε περίπτωση που συμβεί κάποια εξαίρεση κατά το resolve του οργανισμού, τότε η παράμετρος {id} δεν λαμβάνεται καν υπόψιν.

Data Sources:	• Keycloak -> για τους υπαλλήλους
Exceptions:	• **ConflictException** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **NotFoundException** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

Name:	EmployeesView
Roles:	ROLE_ORG_SUPERVISOR
Short Description:	Διαχείριση καταλόγου υπαλλήλων από τους/τις προϊσταμένους/ες κάθε οργανισμού
Base Path:	/organizations/employees/{organization}/{id}
Path Parameters:	• organization (string) -> Το όνομα του οργανισμού Αν δοθεί άλλο όνομα οργανισμού από αυτό στο οποίο εργάζεται ο/η Supervisor τότε επιστρέφεται 403. • id (string) -> Το μοναδικό αναγνωριστικό του υπαλλήλου. Αν δεν βρεθεί το μοναδικό αναγνωριστικό του υπαλλήλου τότε επιστρέφεται 404. Αλλιώς ανοίγει η edit φόρμα για αυτόν. **ΠΡΟΣΟΧΗ**: Ο υπάλληλος πρέπει να ανήκει στον οργανισμό που έχει δοθεί στο {organization} αλλιώς ακόμα και να υπάρχει, θα επιστραφεί 404
Data Sources:	• Keycloak -> για τους υπαλλήλους
Exceptions:	• **ConflictException** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **NotFoundException** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

Name:	ApplicationsView
Roles:	ROLE_ORG_EMPLOYEE
Short Description:	Διαχείριση αιτήσεων ραντεβού για λογαριασμό των οργανισμών που εργάζονται οι υπάλληλοι οργανισμών.
Base Path:	/organizations/applications/{id}
Path Parameters:	• id (String) -> Το μοναδικό αναγνωριστικό της αίτησης Αν δοθεί id το οποίο αντιστοιχεί σε αίτηση, τότε ανοίγει η edit φόρμα.
Data Sources:	• Back-End -> για τις αιτήσεις
Exceptions:	• **HttpClientErrorException.Conflict** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **HttpClientErrorException.Forbidden** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **HttpClientErrorException.NotFound** (η ζητούμενη αίτηση δεν βρέθηκε) -> 404 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

Εξωτερικό Σύστημα

Name:	ApplicationsView
Roles:	ROLE_CITIZEN
Short Description:	Διαχείριση αιτήσεων ραντεβού για τους πολίτες
Base Path:	/applications/{id}
Path Parameters:	• id (String) -> Το μοναδικό αναγνωριστικό της αίτησης Αν δοθεί id το οποίο αντιστοιχεί σε αίτηση, τότε ανοίγει η edit φόρμα.
Data Sources:	• Back-End -> για τις αιτήσεις ραντεβού • Keycloak -> για τους οργανισμούς
Exceptions:	Όσον αφορά το Back-End: • **HttpClientErrorException.Conflict** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **HttpClientErrorException.Forbidden** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **HttpClientErrorException.NotFound** (η ζητούμενη αίτηση δεν βρέθηκε) -> 404 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500

	Όσον αφορά το Keycloak: • **ConflictException** (υπήρξε κάποια σύγκρουση κατά την αποθήκευση του εγγράφου) • **NotFoundException** (το έγγραφο που ζήτησε ο/η υπάλληλος δεν ανήκει στον οργανισμό της) -> 403 • **Exception** (κάτι πήγε πολύ στραβά πχ το Back-End δεν είναι διαθέσιμο) -> 500