**Τεχνική τεκμηρίωση για προγραμματιστές του προγράμματος-πελάτη aadefpa39**

Το έγγραφο αυτό περιγράφει την δομή καθώς και σημαντικά σημεία του κώδικα για το πρόγραμμα-πελάτης aadefpa39 (client απο εδώ και στο εξής). Προορίζεται για προγραμματιστές που πρόκειται να αναπτύξουν/συντηρήσουν/επεκτείνουν τον client και θέλουν να κατανοήσουν την λογική και την ροή εργασίας που έχει ακολουθηθεί. Συμβουλεύοντας το συγκεκριμένο έγγραφο μπορεί κανείς να αποφύγει λάθη, καθώς και να διεκπεραιώσει ταχύτερα απαιτήσεις, οδηγώντας έτσι τον client στην ομαλή ωρίμανση του.

**Εργαλεία και εκδόσεις απαραίτητα για την ανάπτυξη του client**

* IDE: οποιοδήποτε IDE υποστηρίζει Java (προτείνεται τo IntelliJ απο JetBrains)
* Java 1.8 **ΠΡΟΣΟΧΗ:** χρησιμοποιήθηκε κατα την ανάπτυξη η “κανονική” Java και όχι της OpenJDK. Ο λόγος είναι διότι η κανονική έχει bundled την JavaFX, ενώ χρησιμοποιώντας της OpenJDK, είναι απαραίτητο το εξωτερικό OpenJFX project.
* Maven v3: Ο client είναι Maven project, επομένως είναι απαραίτητο για το χτίσιμο, την διαχείρηση εξαρτήσεων και την εγκατάσταση του.
* JFoenix v8.0.8: material UI design για JavaFX. Διαχειρίζεται απο το Maven.
* log4j-api v2.11.1: Διαχειρίζεται απο το Maven.
* fontawesomefx-fontawesome v4.7.0-5: απαραίτητο για την εμφάνιση glyph icons στην εφαρμογή. Διαχειρίζεται απο το Maven, όμως επειδή είναι δεν υπάρχει στο Maven Central, πρεπει ο προγραμματιστης να προσθέσει την παρακάτω ρύθμιση στο αρχείο *.m2/settings.xml*:

**<?xml version="1.0" encoding="UTF-8" ?>**

<settings xsi:schemaLocation='http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd'

xmlns='http://maven.apache.org/SETTINGS/1.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>

<profiles>

<profile>

<repositories>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>bintray-jerady-maven</id>

<name>bintray</name>

<url>https://dl.bintray.com/jerady/maven</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>bintray-jerady-maven</id>

<name>bintray-plugins</name>

<url>https://dl.bintray.com/jerady/maven</url>

</pluginRepository>

</pluginRepositories>

<id>bintray</id>

</profile>

</profiles>

<activeProfiles>

<activeProfile>bintray</activeProfile>

</activeProfiles>

</settings>

* fontawesomefx-commons v8.15: ομοίως με το προηγούμενο.

**Εγκατάσταση του client**

Η εγκατάσταση του προγράμματος πραγματοποιείται μέσω του maven plugin: maven-assembly-plugin, το οποίο δημιουργεί στο path *src\_root/target* το jar *aade39afpa.jar* με όλες τις εξαρτήσεις πακεταρισμένες.

**Δομή αρχείων του κώδικα**

Ο πηγαίος κώδικας του client βρίσκεται έντος του path *src/main*. Στο συγκεκριμένο directory υπάρχουν δύο directories:

* java: περιέχει τον πηγαίο java κώδικα (controllers, μοντέλα, utilities etc.) χωρισμένο σε πακέτα (θα αναλυθεί στην συνέχεια)
* resources: περιέχει τα αρχεία .fxml, το stylesheet του προγραμματος (dark-theme.css), το configuration αρχείο log4j2.xml για τον logger, καθώς και assets της εφαρμογής.

**ΠΡΟΣΟΧΗ**: είναι σημαντικό όλα τα .fxml αρχεία να βρίσκονται στον φάκελο *resources*.

.fxml**αρχεία & controllers**

Για την ανάπτυξη του client επιλέχθηκε η χρήση .fxml αρχείων για το στήσιμο των οθονών UI. Η προσέγγιση αυτή διαχωρίζει κώδικα που αφορά το UI απο business κώδικα ποιυ υπάρχει στους αντίστοιχους controllers. Υπάρχει μια ‘ένα-προς-ένα’ αντιστοιχία για τα αρχεία .fxml και τους controllers. Για κάθε νέα οθόνη (πχ addBuyer.fxml), θα δημιουργείται το αντίστοιχο controller java αρχείο (AddBuyerController.java).

.fxml**αρχεία**

Τα .fxml αρχεία βασίζονται σε xml και παρέχουν την δομή για να χτιστεί το UI. Καθώς οι περισσότερες οθόνες που υπάρχουν ήδη έχουν παρόμοιο στήσιμο, καλό θα ήταν όταν πρόκειται να αναπτυχθεί νέα οθόνη, να αντιγραφεί μια ήδη υπάρχουσα (όπως πχ η getRepresentativeDetails.fxml) και να προσαρμοστεί μέσω αυτής.

Η FXML έχει εμπλουτιστεί με τα material design στοιχεία της βιβλιοθήκης **JFoenix** και στον client χρησιμοποιούνται αυτά. Δείτε [εδώ](http://www.jfoenix.com/documentation.html) τα στοιχεία που παρέχει καθώς και παραδείγματα χρήσης.

**Σημεία προσοχής στα**.fxml**αρχεία**

**Συγκεκριμένα UI στοιχεία & ids για συσχέτιση alert**

Εάν το .fxml αρχείο δεν αναπαριστά κάποιο UI στοιχείο (όπως menu, toolbar, drawer etc.) αλλά αναπαριστά κάποιο βασικό feature container (πχ η οθόνη για την άντληση στοιχείων αντιπροσώπου getRepresentativeDetails.fxml), τότε θα έχει τα δύο αυτά στοιχεία ώς γονείς:

<StackPane fx:id="rootPane" styleClass="root" stylesheets="@dark-theme.css" xmlns="http://javafx.com/javafx/8.0.172-ea"

xmlns:fx="http://javafx.com/fxml/1"

fx:controller="aade.fpa.ui.representative.GetRepresentativeDetailsController">

<children>

<BorderPane fx:id="mainContainer">

<center>

...

...

* **Stackpane** με fx:id="rootPane"" και
* **Borderpane** με fx:id="mainContainer""

Τα συγκεκριμένα ids & στοιχεία είναι σημαντικό να είναι αυτά, διότι με αυτόν τον τρόπο καταλαβαίνουν τα πιθανά alert messages σε ποιό UI στοιχείο γονέα να “πατήσουν”.

Επίσης παρατηρείται πως πρέπει να ορίζεται το αντίστοιχο controller του .fxml αρχείου μέσω της ιδιότητας fx:controller="aade.fpa.ui.representative.GetRepresentativeDetailsController".

**Εξοικονόμηση χώρου**

Καθώς δημιουργείται θέμα χώρου (κυρίως καθετου), τα περιθώρια των στοιχείων θα πρέπει να είναι όσο πιο περιορισμένα γίνεται. Παραδειγματικά:

* αποστάσεις pane απο αριστερα και δεξιά: 10
* αποστάσεις pane απο πάνω και κάτω: 5 - 10
* αποστάσεις στοιχείων εντός των pane αριστερά & δεξιά: 5
* αποστάσεις στοιχείων εντός των pane πάνω & κάτω: 5 - 10

Επίσης η τεχνική των δυναμικών pane, δηλαδή να ανοίγουν οι πίνακες αποτελεσμάτων μόνο όταν υπάρχουν στοιχεία & να κλείνουν ταυτόχρονα τα κριτήρια και αντίστροφα, βοηθούν στην εξοικονόμηση χώρου. Γενικώς, στόχος είναι να μην δημιουργούνται scrollbars (οριζόντια & κάθετα) στην κατάσταση fullscreen & στο default μέγεθος του client.

JFXDatepicker**&**JFXTimepicker

Ένα άλλο σημείο που χρήζει προσοχής είναι στα UI στοιχεία ημερμηνιών & ώρας JFXDatepicker & JFXTimepicker. Καθώς δεν αρκεί η default συμπεριφορά που προσφέρεται απο την JFoenix, έχει επεκταθεί η λειτουργία τους. Απο πλευράς fxml, απαιτείται μόνον στο στοιχείο JFXDatePicker να ορισθεί η ιδιότητα onAction σε συνάρτηση του αντίστοιχου controller που καλεί την AadeUtil.onDateEnter(dateFromField, timeFromField, true); όπως:

**public** **void** **onDateFromEnter**(ActionEvent actionEvent) {

AadeUtil.onDateEnter(dateFromField, timeFromField, **true**);

}

και άρα στο fxml αρχείο θα είναι:

<JFXDatePicker fx:id="dateFromField" promptText="Απο..."

onAction="#onDateFromEnter"/>

Η λογική αυτή χρειάζεται για να ενημερώνεται αυτόματα το αντίστοιχο στοιχείο ώρας σε κάθε αλλαγή ημερομηνίας.

Απο πλευράς controller, απαιτούνται δύο ακόμα ρυθμίσεις για τα στοιχεία JFXDatepicker & JFXTimepicker.

* Ρύθμιση ώρας σε format 24h μέσω (μόνο στα στοιχεία JFXTimepicker): java timeFromField.set24HourView(true);
* Ρύθμιση μετατροπέα που καθαρίζει μη-έγκυρες τιμές (και στα δύο στοιχεία JFXDatepicker & JFXTimepicker): java // Erases storage of previous date/time when current input is invalid dateFromField.setConverter(new SecureLocalDateStringConverter()); timeFromField.setConverter(new SecureLocalTimeStringConverter());

**Custom UI στοιχείο FocusedButton**

Επειδή στις προαπαιτήσεις του client ήταν η δυνατότητα πλοήγησης μέσω TAB, αναπτύχθηκε ένα custom UI στοιχείο, το FocusedButton το οποίο αντικαθιστά το στοιχείο JFXButton της JFoenix. Ο λόγος είναι επειδή το JFXButton δεν εγγυάται πως πατώντας ENTER ύστερα απο πλοήγηση μέσω TAB, θα πατηθεί το κουμπί. Το FocusedButton βρίσκεται στο path */src/java/aade.fpa/util/FocusedButton.java*.

**Controllers**

Οι controllers είναι java αρχεία, το καθένα αντιστοιχεί σε ένα αρχείο .fxml. Οι controllers βρίσκονται στο πακέτο *java/aade.fpa/ui*, όπου εντός αυτού υπάρχει ένας διαχωρισμός κατα features του προγράμματος (πχ buyer, login, supplier, widgets, etc.). Κατα την δημιουργία νέου controller, θα πρέπει να τοποθετείται στο κατάλληλο feature package με το οποίο σχετίζεται, και το όνομά του να αντιστοιχεί με το αρχείο .fxml με το οποίο συσχετίζεται.

Οι controllers περιέχουν τον business κώδικα κάθε οθόνης. Στοιχεία UI συσχετίζονται με κάποιο μοναδικό ID και ενέργειες συσχετίζονται με συγκεκριμένες συναρτήσεις. Αυτό συμβαίνει μέσω του ειδικού annotation @FXML (θα πρέπει φυσικά να αντιστοιχεί το όνομα της μεταβλητής ή συνάρτησης στον controller με αυτό στο αρχείο .fxml).

**Συνάρτηση αρχικοποίησης**initialize()

Στον controller μπορεί να ορισθεί μια συνάρτηση αρχικοποίησης initialize(), η οποία καλείται μια φορά όταν το περιεχόμενο του αντίστοιχου αρχείου .fxml έχουν πλήρως φορτωθεί.

**ΠΡΟΣΟΧΗ**: Ο κλασικός constructor ενός controller **δεν** έχει πρόσβαση στα πεδία με το annotation @FXML, ενώ η initialize() έχει.

Για να ορισθεί η συνάρτηση initialize(), θα πρέπει η κλάση του controller να υλοποιεί το interface Initializable και ύστερα ορίζεται ως εξής:

**@Override**

**public** **void** **initialize**(URL location, ResourceBundle resources) {}

Εντός της συνάρτησς αρχικοποίησης πραγματοποιούνται οποιεσδήποτε απαραίτητες προεργασίες των πεδίων της οθόνης, όπως:

* σετάρισμα bindings java nextPageBtn.disableProperty().bind(Bindings.isEmpty(list));
* αρχικοποίηση ιδιοτήτων πεδίων/στοιχείων java buyerAfmField.setText(UserPrefsUtil.getAfm()); buyerAfmField.setEditable(false);
* χειροκίνητη ρύθμιση τρέχοντος στοιχείου εστίασης java Platform.runLater(() -> repAfmField.requestFocus());
* αρχικοποίηση πινάκων (θα αναλυθεί σε παρακάτω κεφάλαιο)

**Αλλαγή περιεχομένου οθόνης**

Για να πραγματοποιηθεί αλλαγή περιεχομένου οθόνης, χρησιμοποιείται η FXMLLoader loader.load(). Πχ:

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("/addRepresentative.fxml"));

Parent root = **null**;

**try** {

root = loader.load();

} **catch** (IOException e) {

e.printStackTrace();

}

ΑddRepresentativeController addRepresentativeController = loader.getController();

addRepresentativeController.updateModelFromPrevView(r);

((ScrollPane) rootPane.getParent().getParent().getParent()).setContent(root);

VBox.setMargin(root, **new** Insets(0, 0, 0, 30));

Στο παραπάνω παράδειγμα δημιουργείται ένα στιγμιότυπο FXMLLoader. Είναι σημαντικό το path του αρχείου .fxml αρχείου προς φόρτωση να έχει την μορφή όπως το παράδειγμα, δηλαδή "/<όνομα\_αρχείου.fxml>"". Ύστερα καλείται η loader.load.() η οποία φορτώνει την δενδρική δομή .fxml στην μεταβλητή root. Ύστερα δημιουργείται ένα νέο στιγμιότυπο του controller που θέλουμε να φορτώσουμε (στο σημείο αυτό εκτελείται η συνάρτηση initialize() του αντίστοιχου controller του .fxml αρχείου που φορτώθηκε). Φορτώνοντας τον controller ως στιγμιότυπο, μπορούν να εκτελεστούν public μέθοδοι της κλάσης του νέου controller, όπως εδώ όπου περνώντας δεδομένα για τον αντιπρόσωπο r απο την προηγούμενη οθόνη, ενημερώνεται η νέα οθόνη με αυτά μέσω της updateModelFromPrevView(r). Τέλος, είναι σημαντικό να σεταριστεί η νέα fxml στην οθόνη. Αυτό πραγματοποιείται μέσω της x.setContent(root), όπου x ο κόμβος fxml που επιθυμείται να προσαρτιστεί η νέα fxml.

**Κλήσεις SOAP**

O client χρησιμοποεί το πρωτόκολλο SOAP για την επικοινωνία με την υπηρεσία.

***aade.fpa/ws***

Ο φάκελος αυτός περιέχει τον auto-generated κώδικα απο Apache CXF με βάση το WSDL, interceptors, καθώς και config κλάσεις για παραμέτρους της υπηρεσίας.

**Αλλαγές στην υπηρεσία - autogenerated κώδικας μέσω Apache CXF**

Στον φάκελο *src/main/java/aade.fpa/ws* υπάρχει ο auto-generated κώδικας java απο το πρόγραμμα Apache CXF, το οποίο παρέχει τους τύπους/παραμετρους των κλήσεων με βάση το WSDL της υπηρεσίας. Κάθε φορά που γίνεται μια αλλαγή στην υπηρεσία (πχ προσθήκη/αφαίρεση πεδίου/ων ή προσθήκη/αφαίρεση κλήσης/εων), πρέπει να ενημερώνεται το αντίστοιχο σημείο του κώδικα χρησιμοποιώντας το Apache CXF. Δεν προτιμάται η εξ-ολοκλήρου αντικατάσταση του κώδικα με του νέου παραγόμενου, αλλά η επιλεκτική εύρεση εντός του νέου παραγόμενου κώδικα και η αντικατάσταση του συγκεκριμένου σημείου στον ήδη υπάρχον.

***ws/***ServiceManager.java

Η συγκεκριμενη κλάση περιέχει config στοιχεία της υπηρεσίας που μπορούν να ρυθμιστούν, καθώς και μεθόδους ανάκτησης της διεπαφής της υπηρεσίας. Εδώ ορίζονται:

* το namespace του URI της υπηρεσίας "http://vtws39afpa/VtWs39aFPAService"
* το URL του WSDL της υπηρεσίας
  + test: "https://test.gsis.gr/wsaade/VtWs39aFPA/VtWs39aFPA?WSDL"
  + live: "https://www1.gsis.gr/wsaade/VtWs39aFPA/VtWs39aFPA?WSDL"

Οι μέθοδοι ανάκτησης διεπαφής υπηρεσίας χρησιμοποιούνται σε όλες τις κλήσεις SOAP του client.

**Κλάση Interceptor *ws/header/***HeaderHandler.java

Η κλάση αυτή περιέχει την λογική interceptors στα request/response των κλήσεων. Συγκεκριμένα, στην μέθοδο handleMessage() έχουν προστεθεί:

* η λογική της επικάλυψης των στοιχείων του header Username & Password με dummy χαρακτήρες (“ΧΧΧΧΧ”) κατα τα request
* η καταγραφή στον org.apache.logging.log4j.Logger των μηνυμάτων request & response της υπηρεσίας (το parsing των μηνυμάτων πραγματοποιείται μέσω της μεθόδου transformMsgForLog())

**Πραγματοποίηση κλήσης**

Θα αναλυθεί μια τετριμμένη περίπτωση κλήσης, απο την οποία μπορούν να βασιστούν νέες. Οι κλήσεις SOAP πραγματοποιούνται σε νέο thread για να μην μπλοκάρει η ροή του UI. Στην συνέχεια παρουσιάζεται η μέθοδος που καλείται για την αναζήτηση στοιχείων αγοραστή:

**private** **void** **search**() {

String afm = afmField.getText();

Boolean flag = afm.isEmpty();

**if** (flag) {

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Ανεπαρκή δεδομένα", "Παρακαλώ συμπληρώστε τα απαραίτητα πεδία.", **true**, (String) **null**);

afmField.getStyleClass().add("wrong-credentials");

**return**;

}

Task<Vt39AfpaBu3GetBuyerResponseType.Result> task = **new** Task<Vt39AfpaBu3GetBuyerResponseType.Result>() {

**@Override**

**protected** Vt39AfpaBu3GetBuyerResponseType.Result **call**() **throws** Exception {

ObjectFactory objectFactory = **new** ObjectFactory();

VtWs39AfpaBu3InRtType vtWs39AfpaBu3InRtType = objectFactory.createVtWs39AfpaBu3InRtType();

vtWs39AfpaBu3InRtType.setBuyerAfm(objectFactory.createVtWs39AfpaBu3InRtTypeBuyerAfm(afm));

VtWs39AFPAServiceInterface port = ServiceManager.getServiceInterface();

**return** port.vt39AfpaBu3GetBuyer(vtWs39AfpaBu3InRtType);

}

};

**new** Thread(task).start();

task.setOnSucceeded(**new** EventHandler<WorkerStateEvent>() {

**@Override**

**public** **void** **handle**(WorkerStateEvent t) {

**if** (!task.getValue().getVtWs39AfpaBu3ResultRtType().getMessageRec().getMessageCode().isNil()) {

String errorCode = task.getValue().getVtWs39AfpaBu3ResultRtType().getMessageRec().getMessageCode().getValue();

String errorDesc = task.getValue().getVtWs39AfpaBu3ResultRtType().getMessageRec().getMessageDescr().getValue();

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), errorCode, errorDesc, **true**, task.getValue().getVtWs39AfpaBu3ResultRtType().getCallId().getValue().toString());

} **else** {

updateInfo(task.getValue().getVtWs39AfpaBu3ResultRtType());

}

}

});

task.setOnFailed(**new** EventHandler<WorkerStateEvent>() {

**@Override**

**public** **void** **handle**(WorkerStateEvent event) {

String body;

**if** (task.getValue() == **null** && task.getException() != **null**) {

body = task.getException().getMessage();

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Σφάλμα", body, **true**, (String) **null**);

} **else** {

body = "Η διαδικτυακή υπηρεσία δεν είναι προσωρινά διαθέσιμη.";

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Σφάλμα", body, **true**, task.getValue().getVtWs39AfpaBu3ResultRtType().getCallId().getValue().toString());

}

}

});

}

Αρχικά πραγματοποιείται ένα validation ύπαρξης υποχρεωτικών δεδομένων απο το UI. Εάν αποτύχει, εμφανίζεται κατάλληλο alert (θα αναλυθούν τα alert σε επόμενο κεφάλαιο), χρωματίζεται με το error style και επιστρέφει η μέθοδος.

Εάν δεν αποτύχει το validation, υλοποιείται η μέθοδος call() ενός Task (ασύγχρονης εργασίας). Για το Task θα πρέπει να επιλέγεται ο κατάλληλος τύπος επιστροφής απο την υπηρεσία (στην περίπτωση αυτή Vt39AfpaBu3GetBuyerResponseType.Result). Εντός της μεθόδου call() χτίζεται η δομή δεδομένων που θα σταλθεί προς την υπηρεσία με την βοήθεια της κλάσης ObjectFactory. Η ObjectFactory βρίσκεται στον auto-generated κώδικα της Apache CXF και περιέχει βοηθητικές μεθόδους δημιουργίας/σεταρίσματος πεδίων SOAP της υπηρεσίας.

**ΠΡΟΣΟΧΗ:** Είναι μεγάλος ο όγκος των επιλογών στο ObjectFactory. Πρέπει πάντα να επιλέγεται ο κατάλληλος τύπος πεδίου και μεθόδου κατα το χτίσιμο της δομής δεδομένων με βάση την χρήση του. Ο αριθμός της κλήσης είναι καλή ενδεικτική βοήθεια (πχ to **Bu3** στην objectFactory.createVtWs39AfpaBu3InRtType())

Μόλις δημιουργηθεί και σεταριστούν όλα τα απαραίτητα πεδία της δομής δεδομένων προς αποστολή (στην περίπτωση του παραδείγματος, η vtWs39AfpaBu3InRtType), τότε καλείται η μέθοδος ανάκτησης διεπαφής υπηρεσίας της κλάσης ServiceManager.java που περιγράφηκε προηγουμένως για να δημιουργηθεί ένα αντικείμενο διεπαφής υπηρεσίας (στο παράδειγμα εδώ το port). Η μέθοδος call() επιστρέφει καλώντας την κατάλληλη κλήση απο την διεπαφή με περασμένη την σεταρισμένη δομή δεδομένων ως παράμετρο. (στο παράδειγμα return port.vt39AfpaBu3GetBuyer(vtWs39AfpaBu3InRtType);).

Η επόμενη εντολή new Thread(task).start(); εκκινεί την ασύγχρονη κλήση.

Πρέπει φυσικά να οριστούν οι ενέργειες επιτυχίας και αποτυχίας της κλήσης, και αυτά φαίνονται στην συνέχεια του παραδείγματος. Για την περίπτωση επιτυχίας κλήσης (δηλ. το request κατάφερε να φτάσει στην υπηρεσία), πάντα έχει την μορφή όπως η task.setOnSucceeded:

* Ελέγχεται **πάντα πρώτα** η περίπτωση να έχει μήνυμα σφάλματος απο την υπηρεσία (if (!task.getValue().getVtWs39AfpaBu3ResultRtType().getMessageRec().getMessageCode().isNil()) {) και στην περίπτωση που έχει, εμφανίζεται **πάντα** alert με το μήνυμα.
* Αν δεν έχει μήνυμα σφάλματος, μόνο τότε προχωρά σε ενέργειες ενημέρωσης πίνακα/πεδίων κτλ.

Για την περίπτωση αποτυχίας κλήσης, χρησιμοποιείται πάντα ο παρακάτω κώδικας:

task.setOnFailed(**new** EventHandler<WorkerStateEvent>() {

**@Override**

**public** **void** **handle**(WorkerStateEvent event) {

String body;

**if** (task.getValue() == **null** && task.getException() != **null**) {

body = task.getException().getMessage();

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Σφάλμα", body, **true**, (String) **null**);

} **else** {

body = "Η διαδικτυακή υπηρεσία δεν είναι προσωρινά διαθέσιμη.";

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Σφάλμα", body, **true**, task.getValue().getVtWs39AfpaBu3ResultRtType().getCallId().getValue().toString());

}

}

});

ο οποίος ενημερώνει τον χρήστη για την κατάσταση της υπηρεσίας.

**Alert**

Τα alert είναι pop-up dialog boxes που ενημερώνουν τον χρήστη για επιτυχίες/αποτυχίες κλήσεων/ενεργειών, validation σφάλματα κτλ. Μπορουν να είναι και διαδραστικά, δηλαδή να μπορεί ο χρήστης να επιλέξει κάποιο radio button ή κάποιο κουμπί για την συνέχεια της ροής της εργασίας. Η κλάση τους βρίσκεται στο αρχείο *aade.fpa/alert/AlertMaker.java*.

**Τύποι alert**

Εντός της κλάσης AlertMaker.java υπάρχουν πέντε υπογραφές μεθόδων που μπορούν να χρησιμοποιηθούν:

* public static void showMaterialDialog(StackPane root, Node nodeToBeBlurred, List<FocusedButton> controls, String header, String body, boolean isError, String callId): Ο πιο βασικός τύπος alert, δεν κρατάει την απάντηση του χρήστη, απλώς κλείνει με το πάτημα του κουμπιού, χωρις να μπλοκάρει η ροή του προγράμματος (non-blocking). Ως root δίνεται ο κόμβος fxml με το fx:id="root" της οθόνης, ως nodeToBeBlurred δίνεται ο κόμβος fxml fx:id="mainContainer", ως controls δίνεται ένα άδειο ArrayList (συμπληρώνεται εντός της μεθόδου), ως header δίνεται string για την επικεφαλίδα, ως body δίνεται string για το βασικό κείμενο του alert, ως isError δίνεται boolean ανάλογα με το αν πρόκειται για alert σφάλματος ή όχι και τέλος ως callId δίνεται string με τον μοναδικό αριθμό της κλήσης SOAP που σχετίζεται το alert (αν δεν σχετίζεται με κλήση, τότε null).
* public static void showMaterialDialog(StackPane root, Node nodeToBeBlurred, List<FocusedButton> controls, String header, String body, boolean isError, Consumer<Boolean> choice): Τύπος alert που είναι blocking, δηλαδή μέχρι να κλείσει ο χρήστης το alert, η ροή ‘παγώνει’. Αντίστοιχοι παράμετροι με τον πρώτο τύπο, με την διαφορά ότι δεν δέχεται callId και δέχεται Consumer<Boolean> που είναι lambda με την συνέχεια της ροής του προγράμματος.
* public static void showMaterialRadioChoiceDialog(StackPane root, Node nodeToBeBlurred, List<String> choices, String header, String body, Consumer<Integer> choice): Διαδραστικός τύπος alert όπου υπάρχουν radio buttons, όπου με βάση την επιλογή του χρήστη καθορίζεται η συνέχεια της ροής του προγράμματος. Αντίστοιχοι παράμετροι με τον πρώτο τύπο, με την διαφορά ότι αντί για κενό ArrayList πρέπει να περαστεί ArrayList συμπληρωμένο με strings για κάθε radio button. Επίσης δεν υπάρχουν τα isError & callId, ενώ η τελευταία παράμετρος Consumer<Integer> είναι lambda με την επιλογή του χρήστη ώς int.
* public static void showMaterialRadioChoiceDialog(StackPane root, Node nodeToBeBlurred, List<String> choices, String header, String body, String callId, Consumer<Integer> choice): Ίδιος τύπος με τον προηγούμενο, με την διαφορά ότι περνάμε και callId εδώ.
* public static void showMaterialConfirmDialog(StackPane root, Node nodeToBeBlurred, String header, String body, Consumer<Boolean> choice): Διαδραστικός τύπος alert που έχει ένα κουμπί ακύρωσης και ένα επιβεβαίωσης. Ανάλογα με την επιλογή του χρήστη συνεχίζεται η ροή του προγράμματος.

**Παραδείγματα χρήσης alert**

* Για τον πρώτο: AlertMaker.showMaterialDialog(rootPane, mainContainer, new ArrayList<>(), "Σφάλμα", body, true, task.getValue().getVtWs39AfpaSu5ResultRtType().getCallId().getValue().toString());
* Για τον δεύτερο:

AlertMaker.showMaterialDialog((StackPane) root, **null**, **new** ArrayList<>(), "Σφάλμα", body, **true**, choice -> {

FileLockSystem.showAlert = **false**;

((Stage) finalRoot.getScene().getWindow()).close();

});

* Για τον τρίτο:

ArrayList<String> choices = **new** ArrayList<>();

choices.add("Τρέχουσα");

choices.add("Δηλωθείσα");

choices.add("Ακύρωση");

String header = "Αναντιστοιχία ώρας παραστατικού";

String body = "Να χρησιμοποιηθεί ως Ημερομηνία/Ώρα Παραστατικού η τρέχουσα ώρα του συστήματος " + now.format(DateTimeFormatter.ofPattern("HH:mm:ss")) + " ή η δηλωθείσα στη συναλλαγή " + transTimeOnField.getValue().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + ";";

*// Based on user choice, date time is chosen*

AlertMaker.showMaterialRadioChoiceDialog(rootPane, mainContainer, choices, header, body, choice -> {

**if** (choice == 0) {

on[0] = AadeUtil.dateTimeParser(transDateOnField.getValue(), now);

transTimeOnField.setValue(now);

...

...

}

}

* Για τον τέταρτο:

ArrayList<String> choices = **new** ArrayList<>();

choices.add("Ναι");

choices.add("Οχι");

String header = "Επιτυχής ενημέρωση";

body = "Επιτυχής ενημέρωση του αγοραστή με ΑΦΜ: " + task.getValue().getVtWs39AfpaBu1ResultRtType().getBu1OutRec().getBuyerAfm().getValue() + "\nΕπιθυμείτε να ορισθείτε και ως εκπρόσωπος γι’ αυτόν τον ΑΦΜ;";

AlertMaker.showMaterialRadioChoiceDialog(rootPane, mainContainer, choices, header, body, task.getValue().getVtWs39AfpaBu1ResultRtType().getCallId().getValue().toString(), choice -> {

**if** (choice == 0) {

*// Navigate to Add Representative view*

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("/addRepresentative.fxml"));

...

}

...

}

* Για τον πέμπτο:

String confirmhead = "Δημιουργία/ενημέρωση εκπροσώπου";

String confirmbody = "Πρόκειται να δημιουργηθεί/ενημερωθεί εκπρόσωπος";

AlertMaker.showMaterialConfirmDialog(rootPane, mainContainer, confirmhead, confirmbody, choice -> {

**if** (choice) {

....

}

...

}

**Πίνακες**

Υπάρχουν δύο είδη πινάκων στον client:

* Read-only. Απλός πίνακας της JavaFX (TableView). Παράδειγμα στο αρχείο *java/aade.fpa/ui/buyer/GetBuyerHistoryController.java* και *resources/getBuyerHistory.fxml*
* Τροποποιήσιμος πίνακας. Έχει δημιουργηθεί custom κλάση με όλες τις λειτουργίες *java/aade.fpa/widgets/EditableItemTable.java* για τον συγκεκριμένο τύπο πίνακα για λόγους επαναχρησιμοποίησης. Παράδειγμα στο αρχείο *java/aade.fpa/ui/transaction/UpdateTransReprController.java* και *resources/updateTransRepr.fxml*

Οι πίνακες βασίζονται στην τεχνική ‘reflection’, όπου είναι να ανάγκη να υπάρχει ένα μοντέλο το οποίο έχει ήδη ορισθεί, και με βάση τα ονόματα πεδίων του μοντέλου αυτού να γίνει η κατάλληλη συσχέτιση στήλης-πεδίου. Για περισσότερες πληροφορίες σχετικά με την τεχνική αυτή δείτε [εδώ](https://docs.oracle.com/javafx/2/ui_controls/table-view.htm).

**Read-Only πίνακας**

Για τον read-only πίνακα χρειάζεται να ορισθουν οι στήλες και να γίνουν οι αρχικοποιήσεις στο αρχείο το οποίο τοποθετείται. Οι αρχικοποιήσεις γίνονται στην initialize() του controller, συνήθως σε συνάρτηση που ονομάζεται initCols().

**Αρχικοποιήσεις πίνακα**

Οι αρχικοποίησεις που γίνονται σχετίζονται με την λειτουργικότητα που απαιτείται. Θα αναλυθεί ένα παράδειγμα αρχικοποίησης πίνακα του client (GetRepresentativeDetailsController.java), το οποίο έχει τις περισσότερες λειτουργίες ως read-only πίνακας. Παρατίθεται και αναλύεται αποσπασματικά η initCol() του controller του παραδείγματος:

**Reflection**

Εφόσον έχουν δηλωθεί οι στήλες του πίνακα και έχουν συσχετιστεί με αυτές στο αρχείο .fxml, απαιτείται να συσχετιστεί ο πίνακας με το αντίστοιχο πεδίο του μοντέλου. Πχ για το πεδίο buyerAfm: buyerAfmCol.setCellValueFactory(new PropertyValueFactory<>("buyerAfm"));

**Διαχείριση multi-line στηλών με προσθήκη tooltip**

Εάν απαιτηθεί κάποια στήλη με δεδομένα πολλαπλών γραμμών να μετατραπεί σε μιας γραμμής με προσθήκη tooltip τότε χρησιμοποιείται ο ακόλουθος κώδικας (εδω για την στήλη reprNotes):

*// Wrap multline cols - add tooltips to view whole text*

reprNotesCol.setCellFactory(**new** Callback<TableColumn<Representative, String>, TableCell<Representative, String>>() {

**@Override**

**public** TableCell<Representative, String> **call**(TableColumn<Representative, String> p) {

**return** **new** TableCell<Representative, String>() {

**@Override**

**public** **void** **updateItem**(String t, **boolean** empty) {

**super**.updateItem(t, empty);

**if** (t == **null**) {

setTooltip(**null**);

setText(**null**);

} **else** {

Tooltip tooltip = **new** Tooltip();

Representative myModel = getTableView().getItems().get(getTableRow().getIndex());

tooltip.setText(myModel.getReprNotes());

setTooltip(tooltip);

setWrapText(**true**);

setText(t);

}

}

};

}

});

Το παραπάνω snippet πρέπει να προσαρμοστεί για το μοντέλο, το πεδίο και την στήλη.

**Προσθήκη στήλης ενέργειας (**ActionButtonTableCell**)**

Στην περίπτωση που ζητούμενο είναι να προστεθεί στήλη ενέργειας (πχ κουμπί διαγραφής γραμμής) χρησιμοποιείται η custom κλάση *java/aade.fpa/ui/widgets/ActionButtonTableCell.java* ως εξής:

updateCol.setCellFactory(

ActionButtonTableCell.<Representative>forTableColumn("Τροποποίηση εκπροσώπου", **this**::goToUpdateReprView)

);

Στο παράδειγμα προστίθεται μια στήλη ενέργειας τροποποίησης εκπροσώπου. Χρησιμοποιείται η στατική μέθοδος forTableColumn της κλάσης ActionButtonTableCell.java, όπου ως πρώτη παράμετρο δέχεται string με την ετικέτα του κουμπιού και ως δεύτερη παράμετρο δέχεται lamda ή (όπως στο παράδειγμα) method reference που περιγράφει την συμπεριφορά που επιθυμείται.

Φυσικά πρέπει έπειτα η στήλη ενέργειας να προστεθεί στις στήλες του πίνακα: tableView.getColumns().add(updateCol);

**Ενεργοποίηση διπλού κλίκ σε γραμμή πίνακα για ενέργεια**

Για να οριστεί ενέργεια κατα το διπλό κλικ σε γραμμή του πίνακα χρησιμοποιείται κώδικας αντίστοιχος με τον ακόλουθο:

tableView.setRowFactory(tv -> {

TableRow<Representative> row = **new** TableRow<>();

row.setOnMouseClicked(event -> {

**if** (event.getClickCount() == 2 && (!row.isEmpty())) {

goToUpdateReprView(row.getItem());

}

});

**return** row;

});

,όπου εντός της εσωτερικής if η ενέργεια που απαιτείται.

**Αντιγραφή περιεχομένου κελιού με**ctrl+c

**ΠΡΟΣΟΧΗ:** είναι σημαντική η ακόλουθη εντολή για να ενεργοποιηθεί η επιλογή κελιών πίνακα: tableView.getSelectionModel().setCellSelectionEnabled(true);. By default είναι false και επιλέγονται γραμμές, όχι κελιά.

Για να ενεργοποιηθεί η αντιγραφή περιεχομένου κελιού με χρήση ctrl+c χρησιμοποιείται το ακόλουθο snippet:

tableView.setOnKeyPressed(e -> {

**if** (e.getCode() == KeyCode.C && e.isControlDown()) {

AadeUtil.copySelectionToClipboard(tableView);

}

});

**Context menu πίνακα μέσω δεξί κλικ**

Για να ορισθεί context menu του πίνακα μέσω δεξί κλικ, χρησιμοποιείται κώδικας αντίστοιχος με τον ακόλουθο:

**final** ContextMenu contextMenu = **new** ContextMenu();

**final** MenuItem item1 = **new** MenuItem("Αντιγραφή");

**final** MenuItem item2 = **new** MenuItem("Τροποποίηση εκπροσώπου");

**final** MenuItem item3 = **new** MenuItem("Διαγραφή εκπροσώπου");

**final** MenuItem item4 = **new** MenuItem("Κωδικοί μιας χρήσης εκπροσώπου");

**final** SeparatorMenuItem separatorMenuItem = **new** SeparatorMenuItem();

item1.setOnAction(event -> AadeUtil.copySelectionToClipboard(tableView));

item2.setOnAction(event -> goToUpdateReprView(tableView.getSelectionModel().getSelectedItem()));

item3.setOnAction(event -> goToDeleteReprView(tableView.getSelectionModel().getSelectedItem()));

item4.setOnAction(event -> goToOtpView(tableView.getSelectionModel().getSelectedItem()));

contextMenu.getItems().addAll(item1, separatorMenuItem, item2, item4, item3);

tableView.setContextMenu(contextMenu);

Εφόσον δημιουργηθούν τα αντικείμενα ContextMenu, MenuItem και SeparatorMenuItem (αν απαιτηθεί), ορίζονται οι ενέργειες της κάθε επιλογής μέσω της μεθόδου .setOnAction(). Ύστερα προστίθονται οι επιλογές και το διαχωριστικό στο αντικείμενο ContextMenu με την μέθοδο .getItems().addAll()

**ΠΡΟΣΟΧΗ:** η σειρά των αντικειμένων στην μέθοδο προσθήκης έχει σημασία για την σειρά προβολής στο μενού.

Τέλος προστίθεται το ContextMenu στον πίνακα με την εντολή tableView.setContextMenu(contextMenu);.

**ΠΡΟΣΟΧΗ:** για να περιοριστεί η εμφάνιση του μενού μόνο όταν έχει επιλεχθεί κάποιο κελί χρησιμοποιείται ο ακόλουθος κώδικας:

*// Show menu only when something is selected*

tableView.contextMenuProperty().bind(

Bindings.when(Bindings.isNotNull(tableView.getSelectionModel().selectedItemProperty()))

.then(contextMenu)

.otherwise((ContextMenu) **null**));

**Περιορισμός ύψους πίνακα**

Για να περιοριστεί το ύψος του πίνακα, χρησιμοποιώντας εσωτερικό scroll bar ύστερα απο κάποιο πλήθος γραμμών χρησιμοποιείται ο κώδικας:

*// smaller table height, scroll bar for > 10 items*

tableView.setFixedCellSize(23);

tableView.prefHeightProperty().bind(tableView.fixedCellSizeProperty().multiply(10).add(1.01));

tableView.minHeightProperty().bind(tableView.prefHeightProperty());

tableView.maxHeightProperty().bind(tableView.prefHeightProperty());

Το παράδειγμα εμφανίζει εσωτερικό scroll bar στον πίνακα ύστερα απο 10 γραμμές.

**Ενημέρωση πίνακα με αποτελέσματα**

Πέρα απο την μεταβλητή τύπου TableView του καθαυτού πίνακα, ένας controller έχει και μια επιπλέον μεταβλητή τύπου ObservableList που αρχικοποιείται ώς private ObservableList<Representative> list = FXCollections.observableArrayList(); και διατηρεί την κατάσταση του πίνακα. Κάθε φορά που επιθυμείται να ενημερωθεί ο πίνακας, αρχικά γίνονται οι τροποποιήσεις στο ObservableList και ύστερα ενημερώνεται η μεταβλητή τύπου TableView. Για παράδειγμα για την προσθήκη αντιπροσώπων στον πίνακα χρησιμοποιείται το ακόλουθο snippet:

list.clear();

vtWs39AfpaBu7ResultRtType.getBu7OutTab().getItem().forEach(rep -> {

list.add(**new** Representative(

rep.getAa().getValue().longValue(),

rep.getBuyerAfm().getValue(),

rep.getBuyerFullname().getValue(),

...

...

));

});

tableView.getItems().setAll(list);

**EditableItemTable**

Η κλάση EditableItemTable.java (αντίστοιχα fxml αρχείο: *resources/editableItemTable.fxml*) έχει συγκεκριμένες στήλες και έχει επαναχρησιμοποιηθεί σε δύο σημεία του client:

* *java/aade.fpa/ui/transaction/SupplierTransReprController.java*, για καταχώριση συναλλαγών
* *java/aade.fpa/ui/transaction/UpdateTransReprController.java*, για τροποποίηση συναλλαγών

**ΠΡΟΣΟΧΗ:** επειδή έχει επαναχρησιμοποιηθεί σε δύο σημεία, προφανώς χρειάζεται προσοχή και αναλυτικός έλεγχος όταν απαιτείται τροποποίησή του.

Οι αρχικοποίησεις που γίνονται εντός της κλάσης αυτής είναι παρόμοιοι με αυτές που περιγράφηκαν για τον read-only πίνακα, με την εξαίρεση της διαχείρισης ενεργειών τροποποίησης και commit αλλαγών κελιών που είναι συγκεκριμένες για τον editable πίνακα. Επίσης χρησιμοποιείται custom κλάση EditCell ως cell factory των κελιών του editable πίνακα, διότι απαιτήθηκε συγκεκριμένη πλοήγηση (με TAB, SHIFT+TAB) και συμπεριφορά commit που δεν την κάλυπτε η vanilla JavaFX.

Για να τοποθετηθεί ο editable πίνακας σε συγκεκριμένο σημείο, απαιτείται η προσθήκη ενός κενού container (πχ Vbox) όπως για παράδειγμα στο αρχείο supplierTransRepr.fxml:

<VBox fx:id="itemVbox">

<children>

</children>

</VBox>

Ύστερα στον αντίστοιχο controller, προστίθεται το στιγμιότυπο του EditableItemTable στον container:

**try** {

itemTableView = **new** EditableItemTable(itemList, **false**);

itemVbox.getChildren().add(itemTableView);

} **catch** (IOException e) {

e.printStackTrace();

}

Ο constructor της κλάσης EditableItemTable δέχεται δύο παραμέτρους:

* την λίστα τύπου ObservableList που διατηρεί την τρέχουσα κατάσταση του πίνακα
* μια boolean μεταβλητή που δηλώνει αν πρόκειται για τον controller UpdateTransReprController.java ή όχι (για να διαχωριστούν οι λειτουργίες ανάμεσα των οθονών - οι οθόνες χρησιμοποιούν την ίδια κλάση αλλά διαφέρουν στις στήλες)

**Pagination**

Υπάρχουν ορισμένοι πίνακες του client που παρέχουν δυνατότητα σελιδοποίησης. Θα παρατεθεί κώδικας του πίνακα αναζήτησης ιστορικού αγοραστών (*java/aade.fpa/ui/buyer/GetBuyerHistoryController.java*) ο οποίος προσαρμόζεται εύκολα για νέα χρήση καθώς ο κώδικας είναι παρόμοιος σε όλα τα σημεία χρήσης του client:

Αρχικά προστίθονται δύο κουμπιά στο αρχείο fxml:

*<!--Pagination-->*

<HBox alignment="CENTER" prefWidth="200.0" spacing="20.0"

VBox.vgrow="ALWAYS">

<children>

<JFXButton fx:id="prevPageBtn" disable="true" mnemonicParsing="false"

onAction="#onPrevPage" styleClass="prev-page-button">

<font>

<Font size="13.0"/>

</font>

<graphic>

<FontAwesomeIconView glyphName="ARROW\_CIRCLE\_LEFT" size="25"

strokeType="OUTSIDE" strokeWidth="0.0"/>

</graphic>

</JFXButton>

</children>

<JFXButton fx:id="nextPageBtn" alignment="CENTER" defaultButton="true"

mnemonicParsing="false" onAction="#onNextPage"

styleClass="next-page-button">

<font>

<Font size="13.0"/>

</font>

<graphic>

<FontAwesomeIconView glyphName="ARROW\_CIRCLE\_RIGHT" size="25"

strokeType="OUTSIDE" strokeWidth="0.0"/>

</graphic>

</JFXButton>

<VBox.margin>

<Insets left="20.0" right="20.0" top="3.0"/>

</VBox.margin>

</HBox>

Κάθε κουμπί αντιστοιχεί σε μια συνάρτηση του controller (για επόμενη και προηγούμενη σελίδα αντίστοιχα):

**public** **void** **onNextPage**(ActionEvent actionEvent) {

**if** (!list.isEmpty() && currentResult != **null**) {

**boolean** possiblyMoreResults = currentResult.getBu4RowsoutRec().getResultsNo().getValue().longValue()

== currentResult.getBu4RowsoutRec().getFetchSizeApplied().getValue().longValue()

&& (currentResult.getBu4OutTab() != **null**);

**if** (possiblyMoreResults) {

prevPageBtn.setDisable(**false**);

String afm = currentResult.getBu4RowsoutRec().getBuyerAfm().getValue();

Map<String, String> timestampRange = AadeUtil.timestampRangeParser(dateFromField.getValue(), timeFromField.getValue(), dateToField.getValue(), timeToField.getValue());

String sortOrderValue = currentResult.getBu4RowsoutRec().getFetchOrder().getValue();

Integer fetchSize = currentResult.getBu4RowsoutRec().getFetchSizeRequested().getValue() != **null** ? currentResult.getBu4RowsoutRec().getFetchSizeRequested().getValue().intValue() : **null**;

Integer newAaFrom = currentResult.getBu4RowsoutRec().getFetchAaToApplied().getValue().intValue() + 1;

BuyerTaskService buyerTaskService = **new** BuyerTaskService();

Task<Vt39AfpaBu4GetBuyerHistResponseType.Result> task = buyerTaskService.getBuyerHistory(afm, timestampRange, sortOrderValue, fetchSize, newAaFrom);

**new** Thread(task).start();

onSuccessFullTask(task);

} **else** {

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Κανένα αποτέλεσμα", "Δεν βρέθηκαν άλλα στοιχεία", **false**, (String) **null**);

}

} **else** {

AlertMaker.showMaterialDialog(rootPane, mainContainer, **new** ArrayList<>(), "Κανένα αποτέλεσμα", "Δεν βρέθηκαν άλλα στοιχεία", **false**, (String) **null**);

}

}

**public** **void** **onPrevPage**(ActionEvent actionEvent) {

**if** (!list.isEmpty() || currentResult != **null**) {

String afm = currentResult.getBu4RowsoutRec().getBuyerAfm().getValue();

Map<String, String> timestampRange = AadeUtil.timestampRangeParser(dateFromField.getValue(), timeFromField.getValue(), dateToField.getValue(), timeToField.getValue());

String sortOrderValue = currentResult.getBu4RowsoutRec().getFetchOrder().getValue();

Integer fetchSize = currentResult.getBu4RowsoutRec().getFetchSizeRequested().getValue() != **null** ? currentResult.getBu4RowsoutRec().getFetchSizeRequested().getValue().intValue() : **null**;

Integer newAaFrom;

**if** (currentResult.getBu4RowsoutRec().getResultsNo().getValue().intValue() == 0 && !list.isEmpty()) {

newAaFrom = currentResult.getBu4RowsoutRec().getFetchAaFromApplied().getValue().intValue() - currentResult.getBu4RowsoutRec().getFetchSizeApplied().getValue().intValue() \* 2;

} **else** {

newAaFrom = currentResult.getBu4RowsoutRec().getFetchAaFromApplied().getValue().intValue() - currentResult.getBu4RowsoutRec().getFetchSizeApplied().getValue().intValue();

}

**if** (newAaFrom == 1) {

prevPageBtn.setDisable(**true**);

}

BuyerTaskService buyerTaskService = **new** BuyerTaskService();

Task<Vt39AfpaBu4GetBuyerHistResponseType.Result> task = buyerTaskService.getBuyerHistory(afm, timestampRange, sortOrderValue, fetchSize, newAaFrom);

**new** Thread(task).start();

onSuccessFullTask(task);

}

}

Η μεταβλητή currentResult διατηρεί το αποτέλεσμα της τρέχουσας κλήσης αναζήτησης και ενημερώνεται σε κάθε επιτυχημένη κλήση. Στο παράδειγμα: currentResult = task.getValue().getVtWs39AfpaBu4ResultRtType(); στην μέθοδο onSuccessFullTask(). Τα κουμπιά της σελιδοποίησης στέλνουν νέες κλήσεις SOAP με τα τρέχουσα κριτήρια απο currentResult, τροποποιώντας το πεδίο aaFrom καταλλήλως.

**ΠΡΟΣΟΧΗ:** όταν χρησιμοποιείται σελιδοποίηση, κατα την μέθοδο αρχικοποίησης initialize() χρησιμοποιείται η εντολή nextPageBtn.disableProperty().bind(Bindings.isEmpty(list)); για να απενεργοποιείται το κουμπί next όταν είναι άδεια η λίστα του πίνακα. Επίσης πρέπει να απενεργοποιείται το previous στον καθαρισμό: στην μέθοδο onClear(): prevPageBtn.setDisable(true);

**Πακέτο *java/aade.fpa/utils***

Το πακέτο αυτό περιέχει κλάσεις βοηθητικές που δεν συσχετίζονται με κάποιο συγκεκριμένο feature του client, αλλά είναι γενικές.

AadeUtil.java

Η κλάση αυτή περίεχει στατικές μεθόδους γενικές και χρήσιμες καθολικά στην εφαρμογή, όπως parsers, formatters ημερομηνιών & timestamp. Ψάχνοντας τα σημεία χρήσης των μεθόδων σε όλη την εφαρμογή με χρήση εργαλέιου αναζήτησης του IDE, μπορεί κανείς να τις χρησιμοποιήσει για την ανάλογη περίπτωση που χρειάζεται.

FileLockSystem.java

Η κλάση αυτή χρησιμοποιείται απο το αρχείο *java/aade.fpa/main/Main.java* για να αποτρέψει το άνοιγμα δεύτερης διεργασίας του client στο τοπικό μηχάνημα. Χρησιμοποιεί File Channels και Locks για να το επιτύχει.

FloatStringCustomPrecision.java**(Deprecated)**

Η κλάση αυτή είναι **deprecated** και έχει αντικατασταθεί απο την FloatStringTextfieldCustomPrecision.java.

FloatStringTextfieldCustomPrecision.java

Η κλάση αυτή επεκτείνει την FloatStringConverter της Java και χρησιμοποιείται σε πεδία εισαγωγής κειμένου του client όπου εισάγονται float αριθμοί. Διαχειρίζεται το κόμμα και την τελεία ως υποδιαστολή και ορίζει την ακρίβεια σε δύο δεκαδικά ψηφία.

FocusedButton.java

Περιγράφηκε προηγουμένως. Αντικαθιστά το UI στοιχείο JFXButton του JFoenix για να επιτρέπει την επιβεβαιώση κουμπιού με ENTER απο πλοήγηση μέσω TAB.

SecureLocalDateStringConverter.java**&**SecureLocalTimeStringConverter.java

Custom κλάσεις που επεκτείνουν την StringConverter<T> της Java. Χρησιμοποιούνται ως converters στις μεταβλητές των UI στοιχείων JFXDatePicker & JFXTimePicker και καθαρίζουν την προηγούμενη τιμή του πεδίου αν η τρέχουσα είναι μη-έγκυρη.

UserPrefsUtil.java

Κλάση που περιλαμβάνει τις ρυθμίσεις και ιδιότητες σχετικά με τον χρήστη και το proxy στο τοπικό μηχάνημα. Βασίζεται στην κλάση Preferences της Java. Οι ρυθμίσεις αυτές διατηρούνται ανα πολλαπλά στιγμιότυπα του client, αν δεν τις διαγράψει ο χρήστης χειροκίνητα (cache clear), ή μέσω του client.

**ΠΡΟΣΟΧΗ:** οι getters της κλάσης αυτής έχουν συγκεκριμένα default strings που επιστρέφονται σε περίπτωση έλλειψης τιμής και όχι null

***java/aade.fpa/main* & *resources/***main.fxml

Το πακέτο *java/aade.fpa/main* περίεχει την κλάση Main.java η οποία αποτελεί σημείο εκκίνησης της εφαρμογής, καθώς και τον controller MainController.java της οθόνης main.fxml.

Main.java

Η κλάση Main.java αποτελεί σημείο έναρξης της εφαρμογής.

* Θέτει το default locale σε ελληνικά.
* Αν υπάρχουν αποθηκευμένες custom ρυθμίσεις proxy, τις αποθηκεύει και στο config του στιγμιότυπου JVM.
* Ορίζει τις μεταβλητές default μεγέθους οθόνης width & height. (η height απαιτεί ένα offset λόγω ενός σφάλματος στα windows).
* Με βάση το αν υπάρει κωδικός τοπικής εγκατάστασης και αν είναι συνδεδεμένος ο χρήστης, φορτώνει την κατάλληλη οθόνη.
* Τέλος δημιουργεί στιγμοότυπο της κλάσης FileLockSystem.java για την αποτροπή ταυτόχρονων διεργασιών του client.

main.fxml

Αποτελεί βασικό container της κύριας εφαρμογής. Εντός αυτού υπάρχει UI στοιχείο JFXTabPane με τέσσερα tabs (Αγοραστές, Εκπρόσωποι αγοραστών, Έλεγχος, Συναλλαγές). Κάθε tab περιέχει:

* ένα StackPane και εντός κάθε StackPane υπάρχουν:
  + ένα JFXDrawer και
  + ένα ScrollPane

JFXDrawer**του tab**

Κάθε tab περιέχει δικό του ξεχωριστό drawer UI στοιχείο (συρτάρι). Ο λόγος που υλοποιήθηκε με αυτόν τον τρόπο είναι διότι μπορεί μελλοντικά κάθε tab να δείχνει συγκεκριμένες για το tab πληροφορίες και στοιχεία. Ο drawer αρχικοποιείται στις συναρτήσεις initDrawers() & initDrawerController() του MainController.java. Εκεί πραγματοποιούνται οι ρυθμίσεις όπως χρόνος του animation, event handlers και αρχικοποίησεις των controller και fxml αρχεία του drawer.

**ΠΡΟΣΟΧΗ:** αν χρειαστεί νέο tab να έχει drawer, θα ήταν συνετό ο προγραμματιστής να ακολουθήσει τον αντίστοιχο κώδικα όπως και για τα drawers των υπολοιπων tabs.

**ΠΡΟΣΟΧΗ:** μπορεί καθε tab να έχει δικό του *container* για drawer, όμως ο controller και το αντίστοιχο fxml αρχείο για το component είναι **ένα**. (*java/aade.fpa/ui/widgets/DrawerMyReprTransController.java* & *resources/drawerMyReprTrans.fxml*)

Υπάρχει και η συνάρτηση closeDrawers() στο MainController.java το οποίο κλείνει όλα τα ήδη ανοιχτά drawers και εκτελείται κάθε φορά που ο χρήστης επιλέγει κάποιο tab.

ScrollPane**του tab**

Το ScrollPane UI στοιχείο που εμπεριέχει κάθε tab είναι container για την feature-based οθόνη που έχει επιλέξει ο χρήστης απο το μενού δεξιά (toolbar).

Συγκεκριμένα, κάθε φορά που ο χρήστης επιλέγει κάποιο tab, εκτελείται η αντίστοιχη για αυτό το tab μέθοδος του controller MainController.java:

* buyerTabSelected() ή
* repabSelected() ή
* supTabSelected() ή
* transTabSelected(),

οι οποίες έχουν παρόμοια λειτουργία:

* κλείνουν ανοιχτά drawer,
* καλούν την συνάρτηση private void setCentralComponent(String resourceName, TabType tabType), η οποία με βάση τα ορίσματα (όνομα αρχείου fxml feature-based οθόνης για προβολή και enum TabType για το ποίος τύπος tab επιλέχθηκε), θέτει την οθόνη ως περιεχόμενο του κατάλληλου ScrollPane
* ύστερα φορτώνουν και αρχικοποιούν το κατάλληλο toolbar widget ως δεξι στοιχείο του BorderPane του main.fxml container.

**Toolbars**

Τα toolbars είναι τα δεξί μενού που εμφανίζονται στο δεξί μέρος της κύριας οθόνης του client. **Κάθε tab έχει δικό του toolbar component** (δηλαδή ζευγάρι fxml αρχείου & controller). Οι controllers βρίσκονται στο πακέτο *java/aade.fpa/ui/widgets* ενώ τα fxml στα *resources*. Tα toolbar έχουν κουμπιά συσχετισμένα με μεθόδους που φορτώνουν την ανάλογη οθόνη και μεταξύ τους έχουν παρόμοια δομή.

**ΠΡΟΣΟΧΗ:** εάν χρειαστεί να δημιουργηθεί νέο tab, θα ήταν συνετό ο προγραμματιστής να ακολουθούσε την παραπάνω λογική για λόγους συνοχής.

**Χρήσιμα links & resources για JavaFX**

* [The New Boston - Video tutorials (youtube)](https://www.youtube.com/watch?v=FLkOX4Eez6o&list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG)
* [Genuine Coder - Video tutorials (youtube)](https://www.youtube.com/playlist?list=PLhs1urmduZ29jTcE1ca8Z6bZNvH_39ayL)
* [JFoenix Docs](http://jfoenix.com/documentation.html#Download)
* [JavaFx official docs](https://docs.oracle.com/javafx/2/)
* [FontAwesome](https://bitbucket.org/Jerady/fontawesomefx/src/master/) - (glyphicons used in app)
* [SceneBuilder blog tutorial](https://examples.javacodegeeks.com/desktop-java/javafx/scene/javafx-scene-builder-tutorial/)