ДОДАТОК А

Вихідний код програми-сервера

```
/** * Root of Order Aggregate що репрезентують all of business logic of order * and encapsulates all of order state changes. * <p/> * Це помічається змінюваним ордерним штатом, який впливає на бізнес-логіку * і зображення, які також змінюють свій стан і можуть бути тільки accessed через root. * <p/> * OrderRoot reacts on application commands with and responds with events that information of state change * (see examples of event naming {@link com.backgroundknockout.model.message.event}). * <p/> * OrderRoot apply state changes with {@link #apply(com.backgroundknockout.model.message.event.DomainEvent)} * through {@link

com.backgroundknockout.model.impl.domain.MessageHandler} методів. * <p/> * NOTE that you shouldn't place {@code apply()} calls inside {@code com.backgroundknockout.model.impl.domain.MessageHandler} <br/> * methods because it will propagate events outside of OrderRoot on it's reconstruction * (see {@link com.backgroundknockout.model.impl.domain.AbstractDomainCommandHandler#load(com.backgroundknockout.model.impl.message.command.AbstractDomainCommand)} * * @author Olfa Khomyakova *
public classOrderRootextendsAbstractRootimplementsImageContainer {

private finalOrderIdid;
privateOrderStatusstatus;
privateOperatorIdreviewer;
privateInstantreviewStartMoment;

private finalMap<ImageId, Image>images=newHashMap<>();

publicOrderRoot(String domain, OrderId orderId, MessageBus messageBus) {
super(Domain, messageBus);
this.id= orderId; }


voidcreate(finalString customerId,
final booleanprioritized,
finalString txtFile,
finalString workingDir,
finalSet<String> imagePaths) {

finalOrderCreated created =newOrderCreated(domain,id, customerId, prioritized, txtFile, workingDir, imagePaths, ActorImpl.SYSTEM); apply(created);


for(String imagePath : created.getImagePaths()) { ImageId imageId = ImageId.generate(created.entityId(), getFilename(imagePath)); apply( newImageCreated(domain,id, imageId, imagePath, prioritized, created.actor())); } }
```

```
@MessageHandler
public voidhandle(OrderCreated e) {
status=RECEIVED; }


@MessageHandler
public voidhandle(ImageCreated event) { ImageId imageId = event.imageId(); Image
image =

newImage(imageId);
images.put(imageId, image); }


public voidstartReview(Actor actor) {
if(reviewer!=null&& actor.getRole() !=SERVICE_PROVIDER_ADMINISTRATOR) {
throw newIllegalStateException("Order already has reviewer"); } apply(

newOrderReviewStarted(domain,id, actor)); }


@MessageHandler
public voidhandle(OrderReviewStarted e) {
status=IN_REVIEW;
reviewer= OperatorId.fromLoaded(e.actor().getId());reviewStartMoment=
Instant.now(); }public voidsuspend(Actor actor, String notes)
{ reportWorkOnOrderStoppedIfNeeded(actor); apply(

newOrderSuspended(domain,id, notes, actor)); }


public voidsuspendOnTimeout(Actor actor) { checkState(
status==IN_REVIEW,"Order is not allowed to be suspended on timeout in
status "+status); reportWorkOnOrderStoppedIfNeeded(actor); apply(

newOrderSuspended(domain,id,"", actor)); }


@MessageHandler
public voidhandle(OrderSuspended event) {
status=SUSPENDED; }


public voidcompleteReview(Actor actor) { checkState(Iterables.all(
images.values(), inStatus(ImageStatus).REVIEW_COMPLETED)),
"Completing order is not allowed because not all images are 'review
completed'"); reportWorkOnOrderStoppedIfNeeded(actor); apply(

newOrderReviewCompleted(domain,id, actor)); }
```

```
@MessageHandler
public void handle(OrderReviewCompleted event) {
status=REVIEW_COMPLETED; }


public void cancel(Actor actor, String notes) { checkArgument(actor.getRole() ==
SERVICE_PROVIDER_ADMINISTRATOR,"Cancelling order is allowed only to
service admin"); reportWorkOnOrderStoppedIfNeeded(actor); apply(

new OrderCancelled(domain,id, notes, actor)); }


@MessageHandler
public void handle(OrderCancelled event) {
status=CANCELLED;
reviewer= OperatorId.of(event.actor().getId()); }public void archive(Actor actor)
{ checkState(
status.archivePolicy,"Archiving is not allowed in status "+status); apply(
new OrderArchived(domain,id, actor)); }


public void handle(OrderArchived event) { }


public void changeReviewer(Actor actor, OperatorId reviewer)
{ checkArgument(actor.getRole() ==
SERVICE_PROVIDER_ADMINISTRATOR,
"User"+ actor.getId() +"has no permissions to change order reviewer");
checkState(
status.changeReviewerPolicy,"Reviewer assignment is not allowed in status
"+status); apply(
new OrderReviewerChanged(domain,id, Reviewer, actor)); }


@MessageHandler
public void handle(OrderReviewerChanged event) {
reviewer= event.getReviewerId(); }


public void changePriority(Actor actor,boolean prioritized) {checkState(
status.prioritizingAllowed,"Prioritizing is not allowed in status "+status); apply(
new OrderPriorityChanged(domain,id, prioritized, actor)); }


@MessageHandler
public void handle(OrderPriorityChanged event) { }


public void changeDetails(Actor actor, OffsetDateTime dueDate, String notes)
{ checkState(
status.editingAllowed,"Зміна details is not allowed in status "+status); apply(
```

```
newOrderDetailsChanged(domain,id, notes, dueDate, actor)); }


public voidhandle(OrderDetailsChanged event) { }


public voidcompleteImageProcessing(ImageId imageId, ProcessedImageInfo
procInfo, Actor actor) {
finalImage image =images. get (imageId); checkState(image.getStatus() ==
ImageStatus.
IN_PROGRESS,"Can't complete image processing because image isn't in
progress");

if(procInfo.exitCode==RET_VAL_OKAY) { apply(
newImageProcessedSuccessfully(domain,id, imageId, procInfo.exitCode,
procInfo.rotation, procInfo.
defaultProcessing, procInfo.процесуванняDurationMillis, actor)); }
else if(procInfo.exitCode==IMAGE_PROCESSING_SKIPPED) { apply(
newImageSkipped(domain,id, imageId, procInfo.exitCode, procInfo.rotation,
procInfo.
defaultProcessing, procInfo.процесуванняDurationMillis, actor)); }
else if(IsAbleToRepair(procInfo.exitCode)) { apply(
newImageProcessedAndAbleToRepair(domain,id, imageId, procInfo.exitCode,
procInfo.rotation, procInfo.
defaultProcessing, procInfo.процесуванняDurationMillis, actor)); }
else{ apply(
newImageProcessingFailed(domain,id, imageId, procInfo.exitCode,
procInfo.rotation, procInfo.
defaultProcessing, procInfo.процесуванняDurationMillis, actor)); }


if(Iterables.all(images.values(), not(inStatus(ImageStatus).IN_PROGRESS))) {

if(Iterables.any(images.values(), inStatus(ImageStatus).ERROR))) { apply(
newOrderEncounteredError(domain,id, ActorImpl.SYSTEM)); }
else if(Iterables.all(images.values(),
inStatus(ImageStatus).REVIEW_COMPLETED))) { apply(
newOrderReviewCompleted(domain,id, ActorImpl.SYSTEM)); }
else{ apply(
newOrderProcessed(domain,id, ActorImpl.SYSTEM)); } } }



@MessageHandler
public voidhandle(ImageProcessedSuccessfully event) {
finalImage image =images.get(event.imageId()); image.setStatus(ImageStatus.
PENDING_REVIEW); }


@MessageHandler
public voidhandle(ImageProcessedAndAbleToRepair event) {
```

```
finalImage image =images.get(event.imageId()); image.setStatus(ImageStatus.
PENDING_REVIEW); }


@MessageHandler
public voidhandle(ImageSkipped event) {
finalImage image =images.get(event.imageId()); image.setStatus(ImageStatus.
REVIEW_COMPLETED); }


@MessageHandler
public voidhandle(ImageProcessingFailed event) {
finalImage image =images.get(event.imageId()); image.setStatus(ImageStatus.
ERROR); }


@MessageHandler
public voidhandle(OrderProcessed event) {
status=PENDING_REVIEW; }


@MessageHandler
public voidhandle(OrderEncounteredError event) {
status=ERROR; }


public voidrotateImage(ImageId image, ImageRotation rotation, Actor actor)
{ checkArgument(
reviewer.value().equals(actor.getId()),"Image can be rotated only by order
reviewer"); apply(
newImageRotated(domain,id, image, rotation, actor)); }


public voidhandle(ImageRotated event) {
//do nothing, rotation doesn't affect business logic, we don't need to have it in state
}

public voidmarkImageAsLooksGood(ImageId image, Actor actor) {
 apply(newImageMarkedAsLooksGood(domain,id, image, actor)); }


@MessageHandler
public voidhandle(ImageMarkedAsLooksGood event) {
finalImage image =images.get(event.imageId()); image.setStatus(ImageStatus.
REVIEW_COMPLETED); }


public voidmarkImageAsNeedsRepair(ImageId image, Actor actor) { apply(
newImageMarkedAsNeedsRepair(domain,id, image, actor)); }


@MessageHandler
```

```
public void handle(ImageMarkedAsNeedsRepair event) {
final Image image = images.get(event.imageId()); image.setStatus(ImageStatus.
PENDING_REVIEW); }

public void markImageAsNeedsReplacement(ImageId image, Actor actor) { apply(
new ImageMarkedAsNeedsReplacement(domain,id, image, actor)); }

@MessageHandler
public void handle(ImageMarkedAsNeedsReplacement event) {
final Image image = images.get(event.imageId()); image.setStatus(ImageStatus.
ERROR); }

public void replaceImage(ImageId image, Actor actor) { apply(
new ImageSentForReplacing(domain,id, image, actor));
reportStartedProcessingIfNeeded(); }

public void repairImage(ImageId image, List<Tool> toolChain, Actor actor) { apply(
new ImageSentForRepairing(domain,id, image, toolChain, actor));
reportStartedProcessingIfNeeded(); }

public void skipImage(ImageId image, Actor actor) { apply(
new ImageSentForSkipProcessing(domain,id, image, actor));
reportStartedProcessingIfNeeded(); }

public void reprocessImage(ImageId image, Actor actor) { apply(
new ImageSentForReprocess(domain,id, image, actor));
reportStartedProcessingIfNeeded(); }

/** * Handles group of inherited events, which inform that image was sent for
processing */

@MessageHandler
public void handle(ImageSentForProcess event) {
final Image image = images.get(event.imageId()); image.setStatus(ImageStatus.
IN_PROGRESS); }
```

```
@MessageHandler
public void handle(OrderStartedProcessing event) {
status=IN_PROGRESS; }


@Override
public void completeImage(ImageId imageId, Actor actor) {
final ImageStatus currentStatus =images.get(imageId).getStatus();
checkState(currentStatus == ImageStatus.
REVIEW_COMPLETED,
"Image can't be completed when in status "+ currentStatus); apply(
new ImageCompleted(domain,id, imageId, actor));

if(Iterables.all(images.values(), inStatus(ImageStatus).COMPLETED))) { apply(
new OrderCompleted(domain,id, ActorImpl.SYSTEM)); } }



@MessageHandler
public void handle(ImageCompleted event) {
final Image image =images.get(event.imageId()); image.setStatus(ImageStatus.
COMPLETED); }


@MessageHandler
public void handle(OrderCompleted event) {
status=COMPLETED; }


private void reportWorkOnOrderStoppedIfNeeded(Actor actor) {
if(status==IN_REVIEW) {
final Duration workDuration = Duration.between(reviewStartMoment, Instant.now());
apply(new OrderReviewStopped(domain,id,reviewer, workDuration, actor)); } }



private void reportStartedProcessingIfNeeded() {
if(status!=IN_PROGRESS) { apply(
new OrderStartedProcessing(domain,id, ActorImpl.SYSTEM)); } }



@MessageHandler
public void handle(OrderReviewStopped event) {
reviewStartMoment=null; } OrderId getId() {



return id; }
```

```
private booleanisAbleToRepair(ImageExitCode exitCode) {
returnexitCode ==RET_VAL_TOO_MANY_PIXELS_TO_SOLVE; }


enumOrderStatus {

RECEIVED(EditingPolicy.ALLOWED, PrioritizingPolicy.ALLOWED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.DENIED),

IN_PROGRESS(EditingPolicy.ALLOWED, PrioritizingPolicy.ALLOWED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.DENIED),

PENDING_REVIEW(EditingPolicy.ALLOWED, PrioritizingPolicy.ALLOWED,
ChangeReviewerPolicy.
ALLOWED, ArchivePolicy.DENIED),

IN_REVIEW(EditingPolicy.DENIED, PrioritizingPolicy.DENIED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.DENIED),

SUSPENDED(EditingPolicy.DENIED, PrioritizingPolicy.DENIED,
ChangeReviewerPolicy.
ALLOWED, ArchivePolicy.DENIED),

REVIEW_COMPLETED(EditingPolicy.DENIED, PrioritizingPolicy.DENIED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.DENIED),

COMPLETED(EditingPolicy.DENIED, PrioritizingPolicy.DENIED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.ALLOWED),

ERROR(EditingPolicy.ALLOWED, PrioritizingPolicy.ALLOWED,
ChangeReviewerPolicy.
ALLOWED, ArchivePolicy.DENIED),

CANCELLED(EditingPolicy.DENIED, PrioritizingPolicy.DENIED,
ChangeReviewerPolicy.
DENIED, ArchivePolicy.ALLOWED);

public final booleaneditingAllowed;
public final booleanprioritizingAllowed;
public final booleanchangeReviewerPolicy;
public final booleanarchivePolicy; OrderStatus(

booleaneditingPolicy,booleanprioritizingPolicy,booleanchangeReviewerPolicy,bool
eanarchivePolicy) {
this.editingAllowed= editingPolicy;
this.prioritizingAllowed= prioritizingPolicy;
this.changeReviewerPolicy= changeReviewerPolicy;
```

```
this.archivePolicy= archivePolicy; }


private static classPolicy {
public static final booleanALLOWED=true;
public static final booleanDENIED=false; }


private static final classEditingPolicyextendsPolicy {}


private static final classPrioritizingPolicyextendsPolicy {}


private static final classChangeReviewerPolicyextendsPolicy {}


private static final classArchivePolicyextendsPolicy {}}}
```

ДОДАТОК Б

Конфігураційний код для app engine

```
<?xml version= "1.0"encoding= "utf-8"?>
<appengine-web-appxmlns="http://appengine.google.com/ns/1.0">

 <application>backgroundknockout-dev</application>
<version>1</version>

<sessions-enabled>true</sessions-enabled>
<warmup-requests-enabled>true</warmup-requests-enabled>
<threadsafe>true</threadsafe>

<static-files>
 <includepath="/resources/**"expiration="1h">
<http-headername= "Cache-Control"value="private"/>
</include>
</static-files>

<public-root>/resources</public-root>

<resource-files>
<includepath="/WEB-INF/**.*"/>
<includepath="/config.xml"/>
 <!-- Here go resources що є статичними для природи, але повинні бути
доступні за допомогою app code (eg dispatcher servlet handles all of the requests
with context path, як "http://domain/hhcolorlab/pages/index.html" ) -->




<includepath="/resources/pages/index.html"/>
<includepath="/resources/pages/login.html"/>
<includepath="/resources/pages/reset-password.html"/>
</resource-files>

<precompilation-enabled>true</precompilation-enabled>

<system-properties>
<propertyname="java.util.logging.config.file"value="WEB-
INF/logging.properties"/>
<propertyname="backgroundknockout.cloud.temp.result.bucket.prefix"value="
com-backgroundknockout-temp-"/>
<propertyname="backgroundknockout.cloud.result.bucket.prefix"value="com-
backgroundknockout-"/>
<propertyname="backgroundknockout.ftp.ignoredFiles"value=
"db,ini,DS_Store"/>
<propertyname="backgroundknockout.mail.serviceMailSender"value="
backgroundknockout-dev@appspot.gserviceaccount.com "/>
<propertyname="backgroundknockout.mail.orderCreationNotification.recipient
"value= " dmytro.zhykhariev@teamdev.com "/>
```

```
</system-properties>

<inbound-services>
<service>channel_presence</service>
<service>mail</service>
<service>xmpp_message</service>
<service>xmpp_presence</service>
</inbound-services>
</appengine-web-app>


<?xml version= "1.0"encoding= "UTF-8"?>
<cronentries>
<cron>
<url>/admin/cron/orchestrator/orchestrate</url>
<description>Check status regularly</description>
<schedule>every 3 minutes</schedule>
<target>orchestrator-backend</target>
</cron>
<cron>
<url>/admin/cron/orders/archive</url>
<description>Archive all completed or cancelled orders older than [archive
period]</description>
<schedule>every day 00:00 </schedule>
</cron>
<cron>
<url>/admin/cron/orders/suspend</url>
<description>Suspend orders that are not being reviewed or repaired for more than
10 minutes</description>
<schedule>every 2 minutes</schedule>
</cron>
<cron>
<url>/admin/cron/channels/clear</url>
<description>Deletes expired channels</description>
<schedule>every 12 hours</schedule>
</cron>
<cron>
<url>/admin/cron/holidays/update</url>
<description>Update holidays</description>
<schedule>1 of jan 00:00 </schedule>
</cron>
</cronentries>


<queue-entries>
<queue>
<name>default</name>
<rate>50/s</rate>
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
```

```
</queue>
<queue>
<name>general-pull-queue</name>
<mode>pull</mode>
<acl>
<user-email>37440885154-miim9rccsda9vs63f92m7vjmm36hae6p@developer.gserviceaccount.com</user-email>
</acl>
</queue>
<queue>
<name>priority-pull-queue</name>
<mode>pull</mode>
<acl>
<user-email>37440885154-miim9rccsda9vs63f92m7vjmm36hae6p@developer.gserviceaccount.com</user-email>
</acl>
</queue>
<queue>
<name>recalculate-images-pull-queue</name>
<mode>pull</mode>
<acl>
<user-email>37440885154-miim9rccsda9vs63f92m7vjmm36hae6p@developer.gserviceaccount.com</user-email>
</acl>
</queue>
<queue>
<name>image-update</name>
<rate>25/s</rate>
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
</queue>
<queue>
<name>image-copy</name>
<rate>25/s</rate>
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
</queue>
<queue>
<name>image-process</name>
<rate>25/s</rate>
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
</queue>
<queue>
<name>generate-image-preview</name>
<rate>25/s</rate>
```

```
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
</queue>
<queue>
<name>instances-scaling</name>
<rate>5/s</rate>
<retry-parameters>
<task-retry-limit>0</task-retry-limit>
</retry-parameters>
</queue>
<queue>
<name>message-bus</name>
<rate>100/s</rate>
<retry-parameters>
<task-retry-limit>5</task-retry-limit>
</retry-parameters>
</queue>
</queue-entries>


<?xml version= "1.0"encoding= "UTF-8"?>
<configproperties>
 <!-- Configuration of orchestrator -->
<orchestrator-config>
<maximum-number-instances>125</maximum-number-instances>
 <!--<maximum-ave-cpu-load>60</maximum-ave-cpu-load> <minimum-ave-cpu-
load>30</minimum-ave-cpu-load>--> <!- -Task takes 2min that means that 750 tasks
will be processed in a day-->


<maximum-priority-pull-queue-numtasks>0</maximum-priority-pull-queue-
numtasks>
<minimum-priority-pull-queue-numtasks>1</minimum-priority-pull-queue-
numtasks>

<maximum-general-pull-queue-numtasks>0</maximum-general-pull-queue-
numtasks>
<minimum-general-pull-queue-numtasks>1</minimum-general-pull-queue-
numtasks>
 <!--Task takes 2min that means that 30 tasks will be processed in an hour-->
<maximum-recalculate-images-pull-queue-numtasks>0</maximum-recalculate-
images-pull-queue-numtasks>
<minimum-recalculate-images-pull-queue-numtasks>1</minimum-recalculate-
images-pull-queue-numtasks>

<num-instances-to-shut-down>10</num-instances-to-shut-down>
<num-instances-to-create>10</num-instances-to-create>
</orchestrator-config>
 <!-- Configuration of GCE instances -->
<gce-config>
 <!-- Цей проект є тим, що Google Compute Engine є налагодженим і де
```

instances буде створено для виконання вашого застосування. Ви повинні придбати службу з вашого AppEngine orchestrator інструмент, як editor з цього проекту, що він має змогу вгору і вниз. -->

<**projectId**>backgroundknockout-dev</**projectId**>
<!-- Це є місце, де ваш комп'ютер Engine instance буде створений. -->

<**zone**>us-central1-a</**zone**>
<!-- Якщо оркестратор створює нові позначки GCE, повна назва є в цьому форматі: <instancePrefix><number of existing instances + 1> -->

<**instancePrefix**>processor-</**instancePrefix**>
<!-- Prefix for name of the boot disk for new instances. Натиснути на повну назву диска <diskName><instance-name>. -->

<**diskName**>boot-</**diskName**>
<!-- Simple API Access's API key from the project. Це дозволяє вашому застосуванню до використання Compute Engine API. -->

<**projectApiKey**>AIzaSyD6WUGnP_b6bnbXuNwnTAcipnFUdYK33kE</**projectApiKey**><**computeScope**>https://www.googleapis.com/auth/compute</**computeScope**>
<!-- Який зразок розкішного диска, що має всі необхідні softwars, включаючи ваше застосування. Коли новий матеріал є створеним, диск дзвінка буде завантажувати цей шпильку. -->

<**snapshotName**>shutdown-on-timeout-snapshot</**snapshotName**>
<!-- Machine type for the instance. -->

<**machineType**>n1-highcpu-8</**machineType**>
<!-- Startup script to run when the Compute Engine увімкнути starts up. Цей script повинен start your application. Recomend storing startup script в Google Cloud Storage. -->

<**startupScript**>http://commondatastorage.googleapis.com/orchestrator-com-backgroundknockout/startup.sh</**startupScript**>
</**gce-config**>
</**configproperties**>

ДОДАТОК В

Вихідний код програми-клієнта

```
angular.module('orders-front', ['orders-back','images-back','users-back','channel-
back','components','utils','ui.router']) .controller(
'mainOrders',function($scope, $rootScope, $state, $http, $templateCache, Orders, OrderFilter, ImageStatus,
OrderStatus, ChannelService, Users, Roles, LocalStorageUtils) {

// define orders storing variables
$scope.allOrders= [];// orders with any status
$scope.orders= [];// orders with specific statuses to display in table
$scope.orderInReview=null; $scope.
OrderFiltersDefinitions= OrderFilter;// define order filters
$scope.imageStatus= ImageStatus; $scope.
orderStatus= OrderStatus; $scope.
users= []; $scope.
roles= Roles; $scope.

defineFilterApplied=function(Order) {
for(varfilterIdinOrderFilter) {
if(!OrderFilter[filterId])continue;
varfilterDefinition= OrderFilter[filterId];
if(angular.equals(filterDefinition, {status: order.status.value,priority: Order.priority}) ||
angular.equals(filterDefinition, {status: order.status.value})) {
returnfilterId; } } }; $scope.



isJointFilter=function(filterId) {
returnArray.isArray(OrderFilter[filterId.toUpperCase()].status); }; $scope.


filterOrders=function(orders, filterId) {

varfilterDefinition= OrderFilter[filterId];
if(!filterDefinition)returnorders;

varresult= [];

angular.forEach(orders,function(Order) {
varlookupValue;
varmatches;
for(varpropertyinfilterDefinition) {
if(!filterDefinition.hasOwnProperty(property))continue;
lookupValue= filterDefinition[property];
matches= false;
if (lookupValue instanceofArray) {
lookupValue.forEach(function (value) {
if (order[property].value=== value) {
matches = true; } }); }


else{
var orderValue= order[property].value ? order[property].value : order[property];
if (toString(orderValue) === toString(lookupValue)) {
matches = true; } }


if (!matches) {
```

```
break; } }


if (matches) {
this.push(Order); } },

result);

function toString(val) {
return val+''; }


return result; }; $scope.



applyFilter= function (filterId) {$rootScope.
currentFilter= filterId? filterId.toUpperCase() :undefined; $scope.
orders = $scope.filterOrders($scope.allOrders, $rootScope.currentFilter);

var dataTable= $ ("#dt_basic").dataTable();var columns= $rootScope.columns;
if (!dataTable.length|| !columns || !columns.length) return;

var filter= OrderFilter[$rootScope.currentFilter];
for (var i=0; i <columns.length; i++) {
dataTable.fnSetColumnVis(i,!columns[i].isVisible||columns[i].isVisible(filter)); }

dataTable.fnDraw(); };



//filter and show
$scope.showOrders= function (filterId) {$state.
transitionTo('orders.list', {filter: filterId.доLowerCase()}, {reload:false}); };


// define order count function
$scope.getOrdersCount= function (filters) {
return $scope.filterOrders($scope.allOrders, Filters).length; };


// load orders
Orders.list(
function (orders) {

var storedOrder= LocalStorageUtils.getOrderStatus();

if (! $ scope.allOrders) {$scope.
allOrders = []; }

while($scope.allOrders.length >0) {$scope.
allOrders.pop(); }

while(Orders.length > 0) {
var order= orders.pop();
if (storedOrder &&storedOrder.id===order.id) {
order.status.value = storedOrder.status; } $scope.

allOrders.push(order); } $scope.

$$phase|| $scope.$apply(); subscribeChannelService(); $scope.

updateOrderInReview(); LocalStorageUtils.
```

```
clearOrderStatus(); },

function() {
console.debug("list orders fail"); } ); $scope.



isActive=function(filter) {
returnangular.equals($rootScope.currentFilter, Filter); };


functionsubscribeChannelService() { ChannelService.

subscribe(function(change) {$scope.
$broadcast(Зміна.type+ change.operation, change.entity); $scope.
$$phase|| $scope.$apply(); }); $scope.


$on('$destroy',function() { ChannelService.
unsubscribe(); }); } $scope.



$on('OrderUPDATE',function(e, entity) {

if(Roles.isOperator()) {
if((entity.reviewer!=='NO_REVIEWER') && (entity.reviewer!== $rootScope.loggedInUser.id)) {$scope.
allOrders= $.grep($scope.allOrders,function(Order) {
returnorder.id! = entity.id
});
return; } }



varexistingOrder=false; $scope.
allOrders= $.map($scope.allOrders,function(Order) {
if(Order.id== entity.id) {
//if order in not in view by current user
if($scope.orderInReview&& $scope.
orderInReview.id== entity.id&& (entity.
status.value!= OrderStatus.IN_REVIEW|| entity.reviewer!= $rootScope.loggedInUser.id)) {$scope.
orderInReview=null; }


existingOrder=true;
returnentity; }

returnorder; });

if(!existingOrder) {
console.log("--- push order"+ entity.id); $scope.
allOrders.push(entity); }

console.log("--- END"+ entity.id); }); $scope.


$on('OrderDELETE',function(e, entity) {$scope.
allOrders= $.grep($scope.allOrders,function(Order) {returnorder.id! = entity.id}) }); $scope.


getOrderById=function(orders, id) {
varresult= $.grep(orders,function(Order) {
```

```
returnorder.id== id; });

returnresult.length>0?result[0] :null; }; $scope.


expandCollapse=function(){$rootScope.
expanded= !$rootScope.expanded; }; $scope.


defineFilter=function(){$rootScope.
currentFilter= $rootScope.currentFilter.indexOf('PRIORITY')! = -
1?'PRIORITY_ORDERS_PENDING_REVIEW':'ORDERS_PENDING_REVIEW';
return$rootScope.currentFilter.доLowerCase(); }; $rootScope.


$watch('loggedInUser', $scope.updateOrderInReview); $scope.

updateOrderInReview=function() {
if($rootScope.loggedInUser) {$scope.
orderInReview= $.grep($scope.allOrders,function(Order) {
returnorder.reviewer== $rootScope.loggedInUser.id&& order.status.value=='IN_REVIEW'
})[0]; } };



//need to show users names instead of id(email)
Users.list(
function(users) {$scope.
users= users; },

function() {
console.debug("list users fail"); } ); $scope.



getUserNameById=function(id) {
varfoundUser=null; $.
each($scope.users,function(indx, user) {
if(user.id== id) {
foundUser= user;
return false; } });


returnfoundUser?foundUser.firstName+" "+foundUser.lastName: id; }; $scope.


getReviewersData=function(Reviewer) {
varhasReviwer=false;

functiongetUserName(user) {
returnuser.firstName+''+ user.lastName; }


functionsortByName(a, b) {
varaName= getUserName(a);varbName=
getUserName(b);returnaName.доLowerCase() >bName.доLowerCase()?1: -1; }


varusers= $scope.users.slice();
users=users.sort(sortByName);vardata= $.map(users,function(user) {
varisSelected= user.id== reviewer;
if(isSelected) {
hasReviwer=true; }
```

```
returnitem= {
text: getUserName(user),value: user.id,
selected:isSelected
} });


varnoReviewerItem= {
text: $scope.Messages.NO_OPERATOR,
value:'NO_REVIEWER',
selected:!hasReviwer
};
data.unshift(noReviewerItem);
returndata; }; $http.get(


'/pages/common/promt.html', {cache: $templateCache}); $http.get(
'/pages/common/confirm.html', {cache: $templateCache}); }) .controller(

'cancelledOrderMsg',function($rootScope, $scope, $stateParams, $interpolate, Dialog) { $scope.

currentOrderId= $scope.currentOrderId|| $stateParams.id; $scope.
hideCancelBtn=true; $scope.
confirmButtonTitle= $scope.Messages.OK; $rootScope.
activateNotificationDlg=false; $scope.

msg= {
title: $scope.Messages.TITLE_CANCELLED_ORDER,
body: $interpolate($scope.Messages.CONFIRM_CANCELLED_ORDER)({order:
$scope.currentOrderId}) }; Dialog.


show(); $scope.

confirm=function() { Dialog.
hide(); $rootScope.
activateNotificationDlg=true; }; }) .controller(



'operationFailedMsg',function($rootScope, $scope, $stateParams, Dialog) { $scope.

currentOrderId= $scope.currentOrderId|| $stateParams.id; $scope.
hideCancelBtn=true; $rootScope.
activateNotificationDlg=false; $scope.

msg= {
title: $scope.Messages.TITLE_OPERATION_FAILED,
body: $scope.Messages.CONFIRM_OPERATION_FAILED
}; Dialog.

show(); $scope.

confirm=function() { Dialog.
hide(); $rootScope.
activateNotificationDlg=true; }; }) .controller(



'listOrders',function($scope, $rootScope, $filter, $location, $state, $stateParams, Orders, OrderFilter,
OrderStatus, Images, Dialog, Roles) { $rootScope.

expanded=true;
```

```
// define select order handler
$scope.currentOrder=null; $scope.
currentOrderId=null;

if($stateParams.filter){$rootScope.
currentFilter= $stateParams.filter; }
else{!$stateParams.
id&& $location.search({filter: $rootScope.currentFilter.доLowerCase()}); } $scope.


$watchCollection('allOrders',function() {$scope.
applyFilter($rootScope.currentFilter); });


// define convert function
functionformatOffsetDateTime(date) {
return$filter('offsetDateTime')(date); }


functionaddClickHandler(nRow, aData) {
var$nRow= $ (nRow);if($scope.currentOrder&& $scope.currentOrder.id=== aData.id) {
$nRow.addClass("active"); }

$nRow.bind('click',function() {
if($scope.currentOrder&& $scope.currentOrder.id=== aData.id)return;

$nRow.addClass('active').siblings().removeClass('active'); $scope.
currentOrder= aData; $scope.
currentOrderId= $scope.currentOrder.id; $scope.
$$phase|| $scope.$apply(); Images.

list( {
orderId: $scope.currentOrder.id},
function(images) {$scope.
allImages= images; }); });



returnnRow; }


functionisFilterForStatus(filter, status) {
//status is OrderFilter substring, use 'IN_REVIEW' not 'PRIORITY_ORDERS_IN_REVIEW'
returnfilter.status&& (filter.status== status || filter.status.contains&& filter.status.contains(status)); }


functionvalueOrEmpty(property) {
return function(value) {returnvalue[property] ||""}; }


functionsumOfValueOrEmpty(properties) {
return function(value) {
varresult=0;
for(vari=0;i< properties.length;i++) {
result+= value[properties[i]] ||0; }

returnresult||""
}; }


functionlastChangeDate() {
return function(value) {returnvalue? formatOffsetDateTime(value.date) :""}; }
```

```
functiongetLastChangeDateProp() {
return function(obj) {returnobj["lastChange"] ||""}; }


functiongetOperatorsNames(userIds) {
varresult= [];
angular.forEach(userIds,function(userId){ userId &&
result.push($scope.getUserNameById(userId) | userId) });

returnresult.join(","); }


functiongetOperatorColumnName() {
varfilter= OrderFilter[$rootScope.currentFilter.toUpperCase()];
functioncheckStatus(status) {returnisFilterForStatus(filter, Status)}

if(checkStatus(OrderStatus).REVIEW_COMPLETED) || checkStatus(OrderStatus.COMPLETED)) {
return$scope.Messages.COLUMN_COMPLETED_BY_OPERATOR; }

if(checkStatus(OrderStatus).SUSPENDED)) {
return$scope.Messages.COLUMN_OPERATOR; }

if(checkStatus(OrderStatus).CANCELLED)) {
return$scope.Messages.COLUMN_CANCELLED_BY; }

return$scope.Messages.COLUMN_OPERATOR; } $rootScope.


theads= [$scope.
Messages.COLUMN_ORDER_ID, $scope.
Messages.COLUMN_CUSTOMER_NAME, $scope.
Messages.COLUMN_CUSTOMER_REF, $scope.
Messages.COLUMN_NUMBER_OF_IMAGES, $scope.
Messages.COLUMN_NUMBER_OF_ERRORS, $scope.
Messages.COLUMN_PROCESSED_IMAGES, $scope.
Messages.COLUMN_IMAGES_TO_PROCESS, $scope.
Messages.COLUMN_IMAGES_TO_REPAIR, $scope.
Messages.COLUMN_OPERATORS, getOperatorColumnName(), $scope.
Messages.COLUMN_RECEIVED_DATE_TIME, $scope.
Messages.COLUMN_DUE_DATE_TIME, $scope.
Messages.COLUMN_COMPLETED_DATE_TIME, $scope.
Messages.COLUMN_SUSPENDED_DATE_TIME, $scope.
Messages.COLUMN_CANCELLED_DATE_TIME
]; $rootScope.

columns= [ {
"aTargets": [0],"mData":"id","sClass":"dt-id"}, {
"aTargets": [1],"mData":"customerName","sClass":"dt-customerName"}, {
"aTargets": [2],"mData":"customerReference","sClass":"dt-customerReference"}, {
"aTargets": [3],"mData":"numberOfImages","sClass":"dt-img"}, {
"aTargets": [4],"mData":"numberOfImagesOfCriteria","sClass":"dt-
img",isVisible:function(SelectedFilter) {returnisFilterForStatus(ВибраніFilter,'ERROR')},"mRender":
valueOrEmpty('ERROR')}, {
"aTargets": [5],"mData":"numberOfImagesOfCriteria","sClass":"dt-
img",isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'IN_PROGRESS')},"mRender":
sumOfValueOrEmpty(['TO_REPAIR','COMPLETED','REPAIRED','ERROR'])}, {
"aTargets": [6],"mData":"numberOfImagesOfCriteria","sClass":"dt-
img",isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'IN_PROGRESS')},"mRender": valueOrEmpty('TO_PROCESS')},
{
"aTargets": [7],"mData":"numberOfImagesOfCriteria","sClass":"dt-
```

```
img",isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'IN_REVIEW')},"mRender": valueOrEmpty('TO_REPAIR')}, {
"aTargets": [8],"mData":"операторів",isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'REVIEW_COMPLETED') ||
isFilterForStatus(selectedFilter,'COMPLETED')},"mRender": getOperatorsNames}, {"aTargets":
[9],"mData":"Reviewer","sClass":"dt-reviewer",isVisible:function(SelectedFilter) {
returnisFilterForStatus(ВибраніFilter, OrderStatus.REVIEW_COMPLETED) ||
цеFilterForStatus(SelectedFilter, OrderStatus.
COMPLETED) || цеFilterForStatus(SelectedFilter, OrderStatus.
IN_REVIEW) || цеFilterForStatus(SelectedFilter, OrderStatus.
CANCELLED) || цеFilterForStatus(SelectedFilter, OrderStatus.
SUSPENDED) || цеFilterForStatus(SelectedFilter, OrderStatus.
ERROR) || цеFilterForStatus(SelectedFilter, OrderStatus.
PENDING_REVIEW) },
"mRender":function(value) {
varuserName= $scope.getUserNameById(Value);
if(userName==='NO_REVIEWER') {
return$scope.Messages.NO_OPERATOR; }

returnuserName; },

"fnCreatedCell":function(cellNode, value, order) {
varorderStatus= order.status.value;
varisValidOrderStatus=orderStatus=== OrderStatus.SUSPENDED||
orderStatus=== OrderStatus.PENDING_REVIEW||orderStatus=== OrderStatus.ERROR;
if(Roles.isServiceProviderAdmin() &&isValidOrderStatus) { $ (cellNode).
addClass("pseudolink").off('click').on("click",function() {$state.
transitionTo('orders.list.operator', {id: order.id}); }); } } }, {



"aTargets": [10],"mData":"receivedDate","mRender":
formatOffsetDateTime,"sType":'date',"sClass":"dt-date dt-receivedDate"}, {
"aTargets": [11],"mData":"dueDate","mRender": formatOffsetDateTime,"sClass":"dt-date dt-
dueDate","sType":'date'}, {
"aTargets": [12],"mData":"completedDate",isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'COMPLETED') ||
isFilterForStatus(selectedFilter,'REVIEW_COMPLETED')},"mRender":
formatOffsetDateTime,"sType":'date',"sClass":"dt-date dt-completedDate"}, {
"aTargets": [13],"mData": getLastChangeDateProp(),isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'SUSPENDED')},"mRender": lastChangeDate(),"sClass":"dt-date
dt-completedDate"}, {
"aTargets": [14],"mData": getLastChangeDateProp(),isVisible:function(SelectedFilter)
{returnisFilterForStatus(ВибраніFilter,'CANCELLED')},"mRender": lastChangeDate(),"sClass":"dt-date
dt-completedDate"} ];


// define order table options
$rootScope.orderTableOptions= {
aaData: [],
aoColumnDefs: $rootScope.columns,
theads: $rootScope.theads,
"sDom":"<'dt-top-row'>r<'dt-wrapper't><'dt-row dt-bottom-row'<'row'<'col-sm-6'pi><'col-sm- 6 text-
right'l>",
fnRowCallback: addClickHandler,sPaginationType:"two_button",
oLanguage: {
sLengthMenu:"_MENU_ per page",
sInfo:"_START_ - _END_ of _TOTAL_",
sZeroRecords: $scope.Messages.ORDERS_NO_RECORDS,
oPaginate: {
sNext:"<i class='icon-arr-next'></i>",
sPrevious:"<i class='icon-arr-prev'></i>"
```

```
} } }; $(



'#dt_basic_filter').keyup(function() {
vartableElement= $ ('#dt_basic');
if(tableElement) {
vartable=tableElement.dataTable();table.fnFilter($ ("input",this).val()); } }) }) .controller(



'reviewOrderImages',function($rootScope, $scope, $interval, $interpolate, $stateParams, $location, $timeout,
$state, $window, Dialog, OrderStatus, Orders, Images, ImageStatus, ImageType, ImageState, Filters, Utils,
ImageKeyShortcuts, LocalStorage Preloader, ImagesToRepairStore) { $rootScope.expanded =

false; $scope.leaveConfirmed =

false; $scope.currentOrderId = $scope.currentOrderId || $stateParams.id; $scope.ImageState = ImageState;



functionreportActivity(){ Orders.reportActivity({id : $scope.currentOrderId}); }


varhandle = $interval(reportActivity,1*60*1000/*3 minutes*/); reportActivity(); $scope.$on(


'$destroy',function(){ $interval.cancel(handle); });


functionimageLoadWatcher(callback) {
varwatcher = $scope.$watchGroup( [

'model.completedTextures.ORIGINAL',
'model.completedTextures.RESULT',
'model.completedTextures.TRIMAP',
'model.completedTextures.ALPHA'
],function(values) {
varimageCompleted = (function() {
varresult =true; angular.forEach(values,
function(value) { result = result && value !== ImageState.NOT_LOADED; });


returnresult; }());

if(imageCompleted) { watcher(); angular.isFunction(callback) && callback(); } } );



returnwatcher; } imageLoadWatcher(


function() { $timeout(startPreload,
50); });
```

```
functionstartPreload() {
varIMAGES_COUNT_TO_PRELOAD =10;

varnextImagesToPreload = getImagesSetToPreload($scope.selectedImage,
IMAGES_COUNT_TO_PRELOAD); Preloader.addImagesToPreload(nextImagesToPreload); Preloader.load();



varswitchImageWatcher; $scope.$watch(
'filter',function(filter, prevFilter) {
if(filter && filter !== prevFilter) { Preloader.reset(); } }); $scope.$watch(



'selectedImage',function(image, prevImage) {
if(image && prevImage && image.id !== prevImage.id) {// if image changed
angular.isFunction(switchImageWatcher) && switchImageWatcher(); Preloader.pause(); switchImageWatcher =
imageLoadWatcher(

function() { nextImagesToPreload = getImagesSetToPreload(image, IMAGES_COUNT_TO_PRELOAD);
Preloader.addImagesToPreload(nextImagesToPreload,

null,true); Preloader.load(); }); } }); } $scope.currentState = $state.current.name; $scope.disableActions =




false; $scope.model.background = LocalStorageUtils.getBackground() ||
'black'; $scope.switchBackground =

function(bg) { invokeViewportMethod(
'.resulting-side','setBackground', Arguments); $scope.model.background = bg;
LocalStorageUtils.storeBackground(bg); }; $scope.zoomInBoth =



function() { destroySyncZoom(); invokeViewportMethod(

'.original-side, .resulting-side','zoomIn'); setupSyncZoom(); }; $scope.zoomOutBoth =


function() { destroySyncZoom(); invokeViewportMethod(

'.original-side, .resulting-side','zoomOut'); setupSyncZoom(); }; $scope.fitScreenBoth =


function() { destroySyncZoom(); invokeViewportMethod(

'.original-side, .resulting-side','fitScreen'); setupSyncZoom(); };.controller(


'skipImage',function($scope, $stateParams, $state, $interpolate, Images, ImageStatus, Dialog, ExitCode)
{ $scope.

msg= {
title: $scope.Messages.SKIP_IMAGE_TITLE,
body: $interpolate($scope.Messages.SKIP_IMAGE_MSG)({image:name}) }; $scope.
```

```
confirm=function() {$scope.
disableCancelBtn=true; $scope.
disableConfirmBtn=true;

image.status= ImageStatus.IN_PROGRESS;
image.error=false;
image.exitCode= ExitCode.IMAGE_NOT_PROCESSED_YET; Images.

skip({
orderId:orderId,
imageId:imageId
},function() { },

function() {
console.debug("Skip Image operation failed"); $state.
transitionTo('orders.list.error', {id:$scope.currentOrderId}, {reload:false}); }); $scope.


reloadFilterSection(); $scope.
updateCounters(); Dialog.
hide(); }; Dialog.


show();}).controller(
'deleteImage',function($scope, $stateParams, $interpolate, $state, Images, Dialog) {

varorderId= $stateParams.id;
varimageId= $stateParams.image;

varimage= $scope.getImageById(imageId);
varname=image.originalImage?image.originalImage: $stateParams.image; $scope.

msg= {
title: $scope.Messages.DELETE_IMAGE_TITLE,
body: $interpolate($scope.Messages.DELETE_IMAGE_MSG)({image:name}) }; Dialog.


show(); $scope.

confirm=function() {$scope.
disableCancelBtn=true; $scope.
disableConfirmBtn=true; Images.

remove({
orderId:orderId,
imageId:imageId
}, $scope.

allImages= $.grep($scope.allImages,function(image) {return! (image.id==imageId)}); $scope.

reloadFilterSection(); $scope.
updateCounters(); $scope.
checkApprovedStatuses(); Dialog.
hide(); };});



});
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	78 с.
	Інші документи	
2.	Презентаційні матеріали	плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Хомякова О.В.			Розробка web-сервісу з бізнес-логікою з використанням хмарних технологій	Шифр групи	Код напр./спец.
Перевірив		Кудрявцева М.С.				СШІм-22-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					