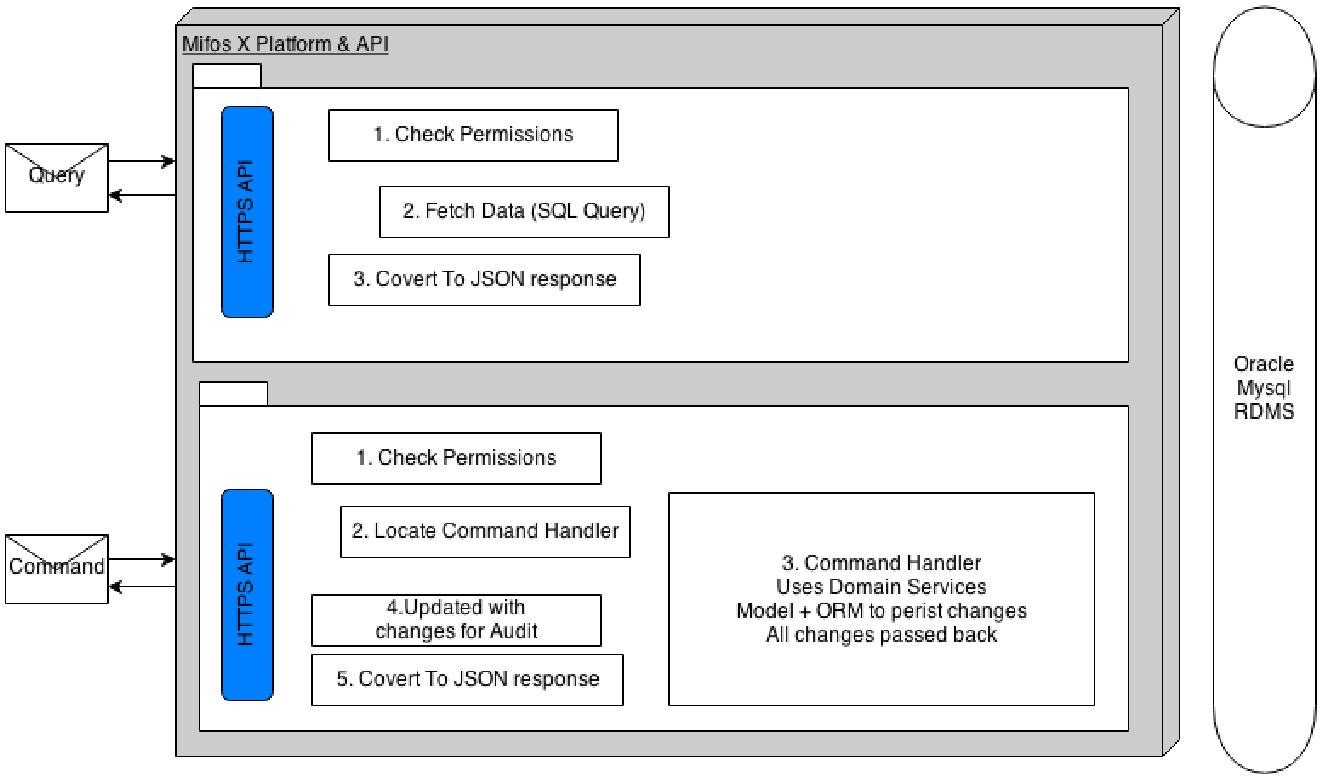
:years: 2015-2022

Design Overview

The implementation of the platform code to process commands through handlers whilst supporting maker-checker and authorisation checks is a little bit convoluted at present and is an area pin-pointed for clean up to make it easier to on board new platform developers. In the mean time below content is used to explain its workings at present.



Mifos X Platform & API
Query
HTTPS API
1. Check Permissions
2. Fetch Data (SQL Query)
3. Covert To JSON response
Command
HTTPS API
1. Check Permissions
2. Locate Command Handler
4.Updated with changes for Audit
5. Covert To JSON response
3. Command Handler Uses Domain Services Model + ORM to perist changes All changes passed back
Oracle Mysql RDMS

Figure 1. CQRS

Taking into account example shown above for the **users** resource.

- Query: GET /users
- HTTPS API: retrieveAll method on **org.apache.fineract.useradministration.api.UsersApiResource** invoked
- UsersApiResource.retrieveAll: Check user has permission to access this resources data.
- UsersApiResource.retrieveAll: Use 'read service' to fetch all users data ('read services' execute simple SQL queries against Database using JDBC)
- UsersApiResource.retrieveAll: Data returned to coverted into JSON response
- Command: POST /users (Note: data passed in request body)
- HTTPS API: create method on org.apache.fineract.useradministration.api.UsersApiResource invoked

UsersApiResource.create

```
    @POST
    @Operation(summary = "Create a User", description = "Adds new application user.\n"
+ "\n"
            + "Note: Password information is not required (or processed). Password
details at present are auto-generated and then sent to the email account given (which
is why it can take a few seconds to complete).\n"
            + "\n" + "Mandatory Fields: \n" + "username, firstname, lastname, email,
officeId, roles, sendPasswordToEmail\n" + "\n"
            + "Optional Fields: \n" +
"staffId,passwordNeverExpires,isSelfServiceUser,clients")
    @RequestBody(required = true, content = @Content(schema = @Schema(implementation =
UsersApiResourceSwagger.PostUsersRequest.class)))
    @ApiResponses({
            @ApiResponse(responseCode = "200", description = "OK", content = @Content
(schema = @Schema(implementation = UsersApiResourceSwagger.PostUsersResponse.class)))
})
    @Consumes({ MediaType.APPLICATION_JSON })
    @Produces({ MediaType.APPLICATION_JSON })
    public String create(@Parameter(hidden = true) final String apiRequestBodyAsJson)
{

        final CommandWrapper commandRequest = new CommandWrapperBuilder() //
                .createUser() //
                .withJson(apiRequestBodyAsJson) //
                .build();

        final CommandProcessingResult result = this.
commandsSourceWritePlatformService.logCommandSource(commandRequest);

        return this.toApiJsonSerializer.serialize(result);
```

Create a CommandWrapper object that represents this create user command and JSON request body. Pass off responsiblity for processing to PortfolioCommandSourceWritePlatformService.logCommandSource

```
        final JsonElement parsedCommand = this.fromApiJsonHelper.parse(json);
        JsonCommand command = JsonCommand.from(json, parsedCommand, this
.fromApiJsonHelper, wrapper.getEntityName(), wrapper.getEntityId(),
                wrapper.getSubentityId(), wrapper.getGroupId(), wrapper.getClientId(),
wrapper.getLoanId(), wrapper.getSavingsId(),
                wrapper.getTransactionId(), wrapper.getHref(), wrapper.getProductId(),
wrapper.getCreditBureauId(),
                wrapper.getOrganisationCreditBureauId(), wrapper.getJobName());

        return this.processAndLogCommandService.executeCommand(wrapper, command,
isApprovedByChecker);
    }

    @Override
    public CommandProcessingResult approveEntry(final Long makerCheckerId) {

        final CommandSource commandSourceInput = validateMakerCheckerTransaction
(makerCheckerId);
        validateIsUpdateAllowed();

        final CommandWrapper wrapper = CommandWrapper.fromExistingCommand
(makerCheckerId, commandSourceInput.getActionName(),
                commandSourceInput.getEntityName(), commandSourceInput.resourceId(),
commandSourceInput.subResourceId(),
                commandSourceInput.getResourceGetUrl(), commandSourceInput
.getProductId(), commandSourceInput.getOfficeId(),
                commandSourceInput.getGroupId(), commandSourceInput.getClientId(),
commandSourceInput.getLoanId(),
                commandSourceInput.getSavingsId(), commandSourceInput.
getTransactionId(), commandSourceInput.getCreditBureauId(),
                commandSourceInput.getOrganisationCreditBureauId(),
commandSourceInput.getIdempotencyKey());
        final JsonElement parsedCommand = this.fromApiJsonHelper.parse
(commandSourceInput.getCommandJson());
        final JsonCommand command = JsonCommand.fromExistingCommand(makerCheckerId,
commandSourceInput.getCommandJson(), parsedCommand,
                this.fromApiJsonHelper, commandSourceInput.getEntityName(),
commandSourceInput.resourceId(),
                commandSourceInput.subResourceId(), commandSourceInput.getGroupId(),
commandSourceInput.getClientId(),
                commandSourceInput.getLoanId(), commandSourceInput.getSavingsId(),
commandSourceInput.getTransactionId(),
                commandSourceInput.getResourceGetUrl(), commandSourceInput
.getProductId(), commandSourceInput.getCreditBureauId(),
                commandSourceInput.getOrganisationCreditBureauId(),
commandSourceInput.getJobName());

        return this.processAndLogCommandService.executeCommand(wrapper, command, true
);
    }

    @Transactional
    @Override
    public Long deleteEntry(final Long makerCheckerId) {

        validateMakerCheckerTransaction(makerCheckerId);
        validateIsUpdateAllowed();

        this.commandSourceRepository.deleteById(makerCheckerId);

        return makerCheckerId;
    }

    private CommandSource validateMakerCheckerTransaction(final Long makerCheckerId) {
        final CommandSource commandSourceInput = this.commandSourceRepository.
findById(makerCheckerId)
                .orElseThrow(() -> new CommandNotFoundException(makerCheckerId));
        if (!commandSourceInput.isMarkedAsAwaitingApproval()) {
            throw new CommandNotAwaitingApprovalException(makerCheckerId);
        }

        this.context.authenticatedUser().validateHasCheckerPermissionTo
(commandSourceInput.getPermissionCode());

        return commandSourceInput;
    }

    private void validateIsUpdateAllowed() {
        this.schedulerJobRunnerReadService.isUpdatesAllowed();
    }
```

Check user has permission for this action. if ok, a) parse the json request body, b) create a JsonCommand object to wrap the command details, c) use CommandProcessingService to handle command

```
    @Override
    @Retry(name = "executeCommand", fallbackMethod = "fallbackExecuteCommand")
    public CommandProcessingResult executeCommand(final CommandWrapper wrapper, final
JsonCommand command,
            final boolean isApprovedByChecker) {
        // Do not store the idempotency key because of the exception handling
        setIdempotencyKeyStoreFlag(false);

        final boolean rollbackTransaction = configurationDomainService
.isMakerCheckerEnabledForTask(wrapper.taskPermissionName());
        String idempotencyKey = idempotencyKeyResolver.resolve(wrapper);
        exceptionWhenTheRequestAlreadyProcessed(wrapper, idempotencyKey);

        // Store idempotency key to the request attribute

        CommandSource savedCommandSource = commandSourceService.saveInitial(wrapper,
command, context.authenticatedUser(wrapper),
                idempotencyKey);
        storeCommandToIdempotentFilter(savedCommandSource);
        setIdempotencyKeyStoreFlag(true);

        final CommandProcessingResult result;
        try {
            result = findCommandHandler(wrapper).processCommand(command);
        } catch (Throwable t) { // NOSONAR
            commandSourceService.saveFailed(commandSourceService.findCommandSource
(wrapper, idempotencyKey));
            publishHookErrorEvent(wrapper, command, t);
            throw t;
        }
        CommandSource initialCommandSource = commandSourceService.findCommandSource
(wrapper, idempotencyKey);
        initialCommandSource.setResult(toApiJsonSerializer.serializeResult(result));
        initialCommandSource.updateResourceId(result.getResourceId());
        initialCommandSource.updateForAudit(result);

        boolean rollBack = (rollbackTransaction || result.isRollbackTransaction()) &&
!isApprovedByChecker;
        if (result.hasChanges() && !rollBack) {
            initialCommandSource.setCommandJson(toApiJsonSerializer.serializeResult
(result.getChanges()));
        }

        initialCommandSource.setStatus(CommandProcessingResultType.PROCESSED.getValue
());
        commandSourceService.saveResult(initialCommandSource);

        if ((rollbackTransaction || result.isRollbackTransaction()) &&
!isApprovedByChecker) {
            /*
             * JournalEntry will generate a new transactionId every time. Updating the
transactionId with old
             * transactionId, because as there are no entries are created with new
transactionId, will throw an error
             * when checker approves the transaction
             */
            initialCommandSource.updateTransaction(command.getTransactionId());
            /*
             * Update CommandSource json data with JsonCommand json data, line 77 and
81 may update the json data
             */
            initialCommandSource.setCommandJson(command.json());
            throw new RollbackTransactionAsCommandIsNotApprovedByCheckerException
(initialCommandSource);
        }
        result.setRollbackTransaction(null);

        publishHookEvent(wrapper.entityName(), wrapper.actionName(), command, result);

        return result;
    }

    private void storeCommandToIdempotentFilter(CommandSource savedCommandSource) {
        if (savedCommandSource.getId() == null) {
            throw new IllegalStateException("Command source not saved");
        }
        saveCommandToRequest(savedCommandSource);
    }
```

if a RollbackTransactionAsCommandIsNotApprovedByCheckerException occurs at this point. The original transaction will of been aborted and we only log an entry for the command in the audit table setting its status as 'Pending'.

- Check that if maker-checker configuration enabled for this action. If yes and this is not a 'checker' approving the command - rollback at the end. We rollback at the end in order to test if the command will pass 'domain validation' which requires commit to database for full check.
- findCommandHandler - Find the correct Hanlder to process this command.
- Process command using handler (In transactional scope).
- CommandSource object created/updated with all details for logging to 'm_portfolio_command_source' table.
- In update scenario, we check to see if there where really any changes/updates. If so only JSON for changes is stored in audit log.