ПРОГРАММА ДЛЯ ЭВМ

Программный комплекс по управлению процессами продажи и оплаты финансовых продуктов

Фрагменты исходного текста программы

Правообладатель-автор(ы): Финтек Системс/Венедиктов Р. А.

2021 г.

Configuration Service

Модуль сохранения структуры BPM-графа в базу данных:

Processengine/PE.GraphEditor.BL/Services/ProcessManager2.cs

```
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using PE.DAL;
using PE.DAL.Model;
using PE.GraphEditor.BL.Exceptions;
using PE.GraphEditor.BL.Interfaces;
using PE.GraphEditor.BL.ModelView;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PE.GraphEditor.BL.Services
{
    public class ProcessManager2 : IProcessManager
    {
        private readonly PEEntities db;
        private readonly ILayoutManager layoutManager;
        private TmplProcess tmplProcess;
        private Dictionary<int, Guid> maplayoutDicInt;
        private Dictionary<Guid, int> maplayoutDicGuid;
        private Dictionary<int, StepEditView> stepsParamsDic;
        private Dictionary<int, TransEditView> transparamsDic;
        private IEnumerable<ITransition> transitions;

        public ProcessManager2(PEEntities pEEntities, ILayoutManager layoutManager)
        {
            this.db = pEEntities;
            this.layoutManager = layoutManager;
        }

        public async Task<(Guid IdProcess, Dictionary<int, Guid> MapGraph)>
SaveAsync(IEnumerable<INode> nodes, IEnumerable<ITransition> transitions, Guid? TemplateId,
IEnumerable<StepEditView> stepParams, IEnumerable<TransEditView> transParamsJSON)
        {
            var maplayout = TemplateId != null ? await layoutManager.LoadMapAsync(TemplateId.Value) :
Enumerable.Empty<(int, Guid)>();

            maplayoutDicInt = maplayout.ToDictionary(g => g.Item1, f => f.Item2);
            maplayoutDicGuid = maplayout.ToDictionary(g => g.Item2, f => f.Item1);

            stepsParamsDic = stepParams.ToDictionary(d => d.GraphId);
            transparamsDic = transParamsJSON.ToDictionary(d => d.TransGraphId);
            this.transitions = transitions;

            tmplProcess = await GetTmplProcessesAsync(TemplateId);

            var dbTrans = await GetAllTransAsync(tmplProcess.Id);

            await DeleteAllTransAsync(dbTrans);

            var dbSteps = (await GetAllStepsAsync(tmplProcess.Id)).ToDictionary(a => a.Id);
            var nodesDic = nodes.ToDictionary(g => g.Id);

            await DeleteStepsFromFromAsync(dbSteps, nodesDic, maplayoutDicGuid);

            await AddorUpdate(nodesDic, dbSteps);

            await SaveTransAsync();

            return (tmplProcess.Id, maplayoutDicInt);
        }
        private async Task AddorUpdate(Dictionary<int, INode> nodesDic, Dictionary<Guid, TmplStep>
dbSteps)
        {
            foreach (var item in nodesDic)
            {
                StepTypeEnum enumStepType = (StepTypeEnum)Enum.Parse(typeof(StepTypeEnum),
item.Value.StepType.Replace("_", ""), true);
                bool tryAdd = false;

                TmplStep dbitem = null;
                if (maplayoutDicInt.TryGetValue(item.Value.Id, out var dbId))
                {
                    dbSteps.TryGetValue(dbId, out dbitem);
                }

                if (dbitem == null)
                {
                    dbitem = new TmplStep() { ProcessTemplateId = tmplProcess.Id, TypeValue =
enumStepType };
                    tryAdd = true;
                }

                if (dbitem.Title != item.Value.Title)
                {
                    dbitem.Title = item.Value.Title;
                }

                if (stepsParamsDic.TryGetValue(item.Value.Id, out var stepViewParam))
                {
                    dbitem.Params = string.IsNullOrWhiteSpace(stepViewParam.Params) ? null :
stepViewParam.Params;
                    switch (enumStepType)
                    {
                        case StepTypeEnum.Task:
                            dbitem.ServiceMethodId = stepViewParam.ServiceMethodId;
                            dbitem.RepeatCount = stepViewParam.RepeatCount == 0 ? null :
stepViewParam.RepeatCount;
                            break;
                        case StepTypeEnum.SubProcess:
                            dbitem.SubprocessTemplateMainId = stepViewParam.SubprocessTemplateMainId;
                            dbitem.RepeatCount = stepViewParam.RepeatCount == 0 ? null :
stepViewParam.RepeatCount;
                            break;

                        case StepTypeEnum.Exit:
                            dbitem.ExitOutCode = stepViewParam.ExitOutCode;
                            break;
                        default:
                            break;
                    }
                }
                else
                {
                    if (db.Entry(dbitem).State == EntityState.Detached)
                    {
                        switch (enumStepType)
                        {
                            case StepTypeEnum.Task:
                            case StepTypeEnum.SubProcess:
                                throw new Exception("Укажите параметры для узла " + item.Value.Title +
" GraphId" + item.Key);
                            case StepTypeEnum.Exit:
                                dbitem.ExitOutCode = 0;
                                break;
                            default:
                                break;
                        }
                    }
                }

                if (db.Entry(dbitem).State == EntityState.Detached)
                {
                    db.TmplSteps.Add(dbitem);
                }

                await db.SaveChangesAsync();

                if (tryAdd)
                {
                    maplayoutDicGuid.Add(dbitem.Id, item.Value.Id);
                    maplayoutDicInt.Add(item.Value.Id, dbitem.Id);
                }
            }
        }

        /// <summary>
        /// Сохраним связи в БД, работает со вместно с данными о связях из localstorage
        /// </summary>
        /// <param name="tmplTransXML"></param>
        /// <returns></returns>
        private Task SaveTransAsync()
        {
            System.Diagnostics.Debug.WriteLine("Количество связей " + transitions.Count());

            foreach (var item in transitions)
            {
                var source = maplayoutDicInt[item.FromId];
                var target = maplayoutDicInt[item.ToId];

                transparamsDic.TryGetValue(item.Id, out var transParam);

                db.TmplTrans.Add(new TmplTran()
                {
                    FromStepId = source,
                    ToStepId = target,
                    OutCode = transParam?.Key
                });
            }
            return db.SaveChangesAsync();
        }

        private Task DeleteStepsFromFromAsync(Dictionary<Guid, TmplStep> dbSteps, Dictionary<int,
INode> nodesDic, Dictionary<Guid, int> maplayoutDicGuid)
        {
            foreach (var item in dbSteps)
            {
                if (maplayoutDicGuid.TryGetValue(item.Value.Id, out var idMap))
                {
                    if (nodesDic.TryGetValue(idMap, out var _))
                    {
                        continue;
                    }
                }
                maplayoutDicInt.Remove(idMap);
                db.Entry(item.Value).State = EntityState.Deleted;
            }

            return db.SaveChangesAsync();
        }

        private async Task DeleteAllTransAsync(TmplTran[] dbTrans)
        {
            foreach (var item in dbTrans)
            {
                db.Entry(item).State = EntityState.Deleted;
            }
            await db.SaveChangesAsync();
        }

        private Task<TmplStep[]> GetAllStepsAsync(Guid id)
        {
            return db.TmplSteps.Where(d => d.ProcessTemplateId == id).ToArrayAsync();
```

```
}

private Task<TmplTran[]> GetAllTransAsync(Guid id)
{
    return db.TmplTrans.Where(k => k.FromStep.ProcessTemplateId == id).ToArrayAsync();
}

public async Task<TmplProcess> GetTmplProcessesAsync(Guid? TemplateId)
{
    TmplProcess tmplProcessResult = null;

    if (TemplateId == null)
    {
        tmplProcessResult = null;
    }
    else
    {
        tmplProcessResult = db.TmplProcesses.FirstOrDefault(k => k.Id == TemplateId);
    }

    if (tmplProcessResult == null)
    {
        var processMain = await db.TmplProcessesMains.AddAsync(new TmplProcessesMain()
        {
            Title = DateTime.Now.ToString(),
            TmplProcesses = new List<TmplProcess>()
            {
                    new TmplProcess()
                    {
                        Version = 1,
                        Title = "Супер пупер процесс",
                        StateValue = TmplProcessStateEnum.Draft,
                        Id = TemplateId ?? Guid.NewGuid()
                    },
            },
        });

        await db.SaveChangesAsync();

        tmplProcessResult = processMain.Entity.TmplProcesses.First();
    }
    else
    {
        if (tmplProcessResult.StateValue != TmplProcessStateEnum.Draft)
        {
            var tmplProcessResultNew = new TmplProcess()
            {
                Version = tmplProcessResult.Version,
                Title = tmplProcessResult.Title,
                StateValue = TmplProcessStateEnum.Draft,
                ProcessTemplateMainId = tmplProcessResult.ProcessTemplateMainId,
            };

            await db.TmplProcesses.AddAsync(tmplProcessResultNew);
            await db.SaveChangesAsync();

            await CopyStepstoParamsAsync(tmplProcessResult.Id);
            await CopyTranstoParamsAsync(tmplProcessResult.Id);

            maplayoutDicGuid.Clear();
            maplayoutDicInt.Clear();

            return tmplProcessResultNew;
        }
        else
        {
            await CopyTranstoParamsAsync(tmplProcessResult.Id);
        }
    }
    return tmplProcessResult;
}
```

```
        private async Task CopyTranstoParamsAsync(Guid id)
        {
            var oldTrans = await db.TmplTrans.Where(k => k.FromStep.ProcessTemplateId ==
id).AsNoTracking().ToArrayAsync();

            foreach (var item in oldTrans)
            {
                if (maplayoutDicGuid.TryGetValue(item.FromStepId, out var mapFromId) &&
maplayoutDicGuid.TryGetValue(item.ToStepId, out var mapToId))
                {
                    var trans = transitions.FirstOrDefault(g => g.FromId == mapFromId && g.ToId ==
mapToId);
                    if (trans != null)
                    {
                        if (transparamsDic.TryGetValue(trans.Id, out var _))
                        {
                            continue;
                        }

                        transparamsDic.Add(trans.Id, new TransEditView()
                        {
                            Key = item.OutCode,
                            TransGraphId = trans.Id,
                        });
                    }
                }
                else
                {
                    throw new NotFoundTransinGraph("Не найден индефикатор связи графа mapFromId " +
item.FromStepId + " mapToId " + item.ToStepId);
                }
            }
        }

        private async Task CopyStepstoParamsAsync(Guid id)
        {
            var oldSteps = await db.TmplSteps.Where(k => k.ProcessTemplateId ==
id).AsNoTracking().ToListAsync();

            foreach (var item in oldSteps)
            {
                if (maplayoutDicGuid.TryGetValue(item.Id, out var mapGraphId))
                {
                    if (stepsParamsDic.TryGetValue(mapGraphId, out var _))
                    {
                        continue;
                    }
                    stepsParamsDic.Add(mapGraphId, new StepEditView()
                    {
                        ExitOutCode = item.ExitOutCode,
                        Params = item.Params,
                        RepeatCount = item.RepeatCount,
                        ServiceMethodId = item.ServiceMethodId,
                        SubprocessTemplateMainId = item.SubprocessTemplateMainId,
                        TypeId = item.TypeId,
                        Title = item.Title,
                        GraphId = mapGraphId,
                    });
                }
                else
                {
                    throw new NotFoundStepinGraph("Не найден индефикатор шага");
                }
            }
        }
    }
}
```

Metadata Service

Фрагмент модуля генерации запроса на основе метаданных

Applicationrepository/DS.AppRepository.Core/Data/DataManager.cs

```
        /// <summary>
        /// Read many records from <paramref name="table"/>.
        /// if need to read children tables, then host table and child tables fields must include id
and parent_id columns
        /// </summary>
        /// <param name="conn"></param>
        /// <param name="trans"></param>
        /// <param name="table">information about table</param>
        /// <param name="fields">fields information</param>
        /// <param name="options">Read options</param>
        /// <returns></returns>
        /// <exception cref="UnknownColumnException">
        /// 1. Filter clause include unknown field
        /// 2. Include unknown child column
        /// </exception>
        /// <exception cref="InvalidDataFieldTypeException">Try to include column whitch type are not
Children</exception>
        public async Task<List<JObject>> ReadManyAsync(
            DbConnection conn,
            DbTransaction trans,
            IDBTableInfo table,
            ReadOptions options = null,
            FilterClause filter = null,
            IEnumerable<OrderClause> order = null,
            IDataContext dataContext = null)
        {
            var dbpair = new DbPair(conn, trans);

            //get fields for main table
            var tableFields = await _fieldsSource.FieldsAsync(conn, trans, table);
//fields[table.TableId];

            SqlBuilder builder = new SqlBuilder();

            TableAliasMaker dictAliasMaker = new TableAliasMaker();

            //make select clause for main table
            builder.FieldsClause(tableFields, dictAliasMaker: dictAliasMaker);

            IDictionary<string, IEnumerable<string>> nextIncludes = null;
            var includes = options?.Include.SplitIncludes(out nextIncludes);
            (var includeReferences, var includeChildren) = includes.SplitIncludesByFieldType(table,
tableFields);

            if ((options?.ForbidIncludeChildren ?? false) && (includeChildren?.Any() ?? false))
                throw new NotSupportedException("Include children tables was forbidden");

            //make left join to references
            IEnumerable<(string path, IDBTableInfo table, IEnumerable<KeyValuePair<string,
IDBFieldInfo>> children)> nestedChildren = null;
            if (includeReferences != null)
                nestedChildren = await IncludeReferencesFieldsAsync(conn, trans, builder, table,
dictAliasMaker, includeReferences,
                    nextIncludes, QueryBuilder.MAIN_TABLE_ALIAS, null);

            //make where clause if filter exists
            IEnumerable<FilterParameterInfo> filterParamsInfo = null;
            if (filter != null)
                filterParamsInfo = await MakeWhereClauseAsync(dbpair, builder, _filterCompiler,
tableFields, filter, dictAliasMaker, dataContext);

            if (order != null)
                await MakeOrderClauseAsync(dbpair, builder, tableFields, dictAliasMaker, order);
            else if (options?.Count != null || options?.Offset != null)
                builder.OrderBy(new OrderClause { Field = QueryBuilder.MAIN_TABLE_FIELD_PREFIX +
IdField });
```

```
            string sql = builder.DefaultTemplate(table.TableName, options?.Count,
options?.Offset).RawSql;

            // prepare filter params
            IEnumerable<KeyValuePair<string, object>> sqlParams = null;
            if (filterParamsInfo != null)
                sqlParams = filterParamsInfo.ToSqlParams();

            //create filler for main table
            var filler = new JsonReferenceFiller();//JsonReferenceFiller(mainss);

            List<JObject> list = await ExecReadManyAsync(conn, trans, sql, filler,
sqlParams?.Cast<object>().ToArray());

            //if need read relatives tables
            if (list.Count > 0 && (includeChildren?.Any() ?? false))
                await PopulateAllChildrenFieldsAsync(conn, trans, table, list, options,
includeChildren, nextIncludes);

            if (nestedChildren != null) {
                var nestedOpstions = new ReadOptions { };
                foreach (var child in nestedChildren) {
                    var nestedObjects = list.Select(i => i.SelectToken(child.path)).Where(i => i !=
null && i.Type != JTokenType.Null).Cast<JObject>().ToList();
                    await PopulateAllChildrenFieldsAsync(conn, trans, child.table, nestedObjects,
nestedOpstions, child.children, null);
                }
            }

            return list;
        }

        /// <summary>
        /// Request all childs table from DB and add child fields to records in <paramref
name="hostRecords"/>
        /// </summary>
        /// <param name="conn"></param>
        /// <param name="trans"></param>
        /// <param name="table">information about host table</param>
        /// <param name="fields">information about fields</param>
        /// <param name="hostRecords">list of records in host table</param>
        /// <exception cref="UnknownColumnException">Try to include Unknown chinldren column
</exception>
        /// <exception cref="InvalidDataFieldTypeException">Try to include column whitch type are not
Children</exception>
        private async Task PopulateAllChildrenFieldsAsync(
            DbConnection conn,
            DbTransaction trans,
            IDBTableInfo table,
            List<JObject> hostRecords,
            ReadOptions options,
            IEnumerable<KeyValuePair<string, IDBFieldInfo>> includes,
            IDictionary<string, IEnumerable<string>> nextIncludes)
        {
            //TODO: check fields contains id field and add it if missed?

            //make dictionary for quick access to host records by id
            Dictionary<JValue, JObject> hostRecordsDict = hostRecords
                .ToDictionary(i => i[TableInfoDefinitions.PrimaryKeyField] as JValue);

            JArray hostRecordsIds = new JArray();
            foreach (var id in hostRecordsDict.Keys)
                hostRecordsIds.Add(id);

            //all fields from main table
            IFieldsCollection tableFields = await _fieldsSource.FieldsAsync(conn, trans, table);//
fields[table.TableId];

            //get serialization settings for main table
            //ISerializationSettings mainss = ssb?[table.TableId];
            IEnumerable<IDBFieldInfo> childFields = includes.Select(i => i.Value);

            //enumerate children fields and perform request for each field
            foreach (var field in childFields) {
                //if (field.IsReferencedObjectIndependent == false) {
                IEnumerable<string> childIncludes = null;
                string propName = field.DbName;
                nextIncludes?.TryGetValue(propName, out childIncludes);
                await PopulateOneChildrenFieldAsync(conn, trans, hostRecordsIds, hostRecordsDict,
field, childIncludes);
                //}
            }
        }

        /// <summary>
        /// Request one child table from <paramref name="hostField"/> from DB
        /// and add one child field to records in <paramref name="hostRecords"/>
        /// </summary>
        /// <param name="conn"></param>
        /// <param name="trans"></param>
        /// <param name="fields">information about fields</param>
        /// <param name="hostRecords">dictionary with records from host table</param>
        /// <param name="hostField">field in host records to populate</param>
        private async Task PopulateOneChildrenFieldAsync(
            DbConnection conn,
            DbTransaction trans,
            JArray hostRecordsIds,
            Dictionary<JValue, JObject> hostRecords,
            IDBFieldInfo hostField,
            IEnumerable<string> currentIncludes)
        {
            IDBTableInfo table = hostField.ToTableInfo();
            var options = new ReadOptions() {
                Include = currentIncludes,
                ForbidIncludeChildren = true //forbid to include children tables from another children
tables
            };

            //create filter clause, all foreign keys contains in list of id
            FilterClause filter = FilterClause.CreateCondition(SqlOperandsConditionMaker.ContainsIn,
                hostField.ReferencedFieldName, hostRecordsIds);
            List<JObject> childRecords = await ReadManyAsync(conn, trans, table, options, filter, null,
null);

            //field name in host record
            string fieldName = hostField.DbName;// ssb.JsonName(hostField.TableId, hostField.DbName);

            //add empty field to all host records.
            //Because if omit this, then record without child records will not be include to object
            foreach (var pair in hostRecords)
                pair.Value[fieldName] = new JArray();

            //ISerializationSettings childss = ssb?[table.TableId];

            //enumerate all childs records and add it to host record
            foreach (var child in childRecords) {
                JValue hostId = (JValue)child[hostField.ReferencedFieldName];
                if (!hostRecords[hostId].TryGetValue(fieldName, out JToken jtoken)) {
                    JArray array = new JArray { child };
                    hostRecords[hostId].Add(fieldName, array);
                } else
                    ((JArray)jtoken).Add(child);
            }
        }
```

Interface Service

Модуль контроллера процесса верификации:

Visualfronts/VisualFrontsWebCore/Controllers/VerificationController.cs

```
using System;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using VF.BL.Abstractions.Sources;
using VF.BL.Model.Verification;
using VF.Frontend.Web.ExceptionFilter;
using VF.Frontend.Web.Models;
using VF.Frontend.Web.Models.VerificationScreen;
using VF.Frontend.Web.Settings.Interfaces;
using VF.Frontend.Web.Settings.Verification.Check;

namespace VF.Frontend.Web.Controllers
{
    public class VerificationController : Controller
    {
        private readonly IVerificationSource _verificationSource;
        private readonly IVerificationCheck _verificationCheck;
        private readonly IPageTemplateSource _pageTemplateSource;
        private readonly IUserSettingsSource _userSettingsSource;
        private readonly ILogger<VerificationController> _logger;
        private const int DefaultPhotoHeight = 20;

        public VerificationController(IVerificationSource verificationSource
            , IVerificationCheck verificationCheck
            , IUserSettingsSource userSettingsSource
            , ILogger<VerificationController> logger
            , IPageTemplateSource pageTemplateSource)
        {
            _verificationSource = verificationSource;
            _verificationCheck = verificationCheck;
            _pageTemplateSource = pageTemplateSource;
            _userSettingsSource = userSettingsSource;
            _logger = logger;
        }

        [Route("/page/{pageName}/question/{divId}/record/{recordId}")]
        [HttpPost]
        [TypeFilter(typeof(ExceptionPageFilter))]
        public async Task<IActionResult> PartialLoad(string divId, Guid recordId, string pageName, bool 
isUpdateSession = true)
        {
            VerificationScreenViewModel result;

            result = await GetVerificationScreenViewModel(pageName, divId, recordId, isUpdateSession);

            var (isValid, errors) = _verificationCheck.ValidateChildren(result.VerificationScreen);
            foreach (var (fieldName, errorMessage) in errors)
            {
                ModelState.AddModelError(fieldName, 
errorMessage.GetValue(CultureInfo.CurrentUICulture.Name).ToString());
            }

            return PartialView("Question", result);
        }

        /// <summary>
        /// Проверка секретного кода
        /// </summary>
        /// <param name="model">модель запроса</param>
        /// <returns>успех или нет</returns>
        [HttpPost]
        public bool ChekSecretCode(CheckCodeAnswerModel model)
        {
            var sourceSecretCode = _userSettingsSource.GetSecretCode(model.AnswerProcedureId,
model.QuestionId);
            var result = sourceSecretCode.Equals(model.SecretCode);

            _logger.LogInformation(@"Chek verification Secret Code answer model={@model},
result={result}", model, result);

            return result;
        }

        [HttpPost]
        public async Task<IActionResult> SendAnswers([FromForm] VerificationQuestionSaveAnswerModel
request)
        {
            ModelState.Clear(); // иначе сохраняются ошибки, которые были
            var isUpdateSession = false;
            request.Answers = request.Answers.Where(w => w.QuestionId > 0).ToArray();
            var (isValid, errors) = _verificationCheck.Validate(request);

            if (!isValid)
            {
                foreach (var (fieldName, errorMessage) in errors)
                {
                    ModelState.AddModelError(fieldName,
errorMessage.GetValue(CultureInfo.CurrentUICulture.Name).ToString());
                }
            }

            var secretAnswers = request.Answers.Where(w => w.IsSecret).ToList();

            foreach (var answer in secretAnswers.Where(s => string.IsNullOrEmpty(s.SecretCheckValue)))
            {
                ModelState.AddModelError($@"Answers[{answer.ArrayId}].SecretValueError",
ErrorMessage.SecretCodeNotChek.GetValue(CultureInfo.CurrentUICulture.Name)?.ToString() ??
String.Empty);
            }

            foreach (var answer in secretAnswers)
            {
                var sessionSecretCode =
                    _userSettingsSource.GetSecretCode(request.AnswerProcedureId, answer.QuestionId);

                if (string.IsNullOrEmpty(sessionSecretCode))
                {
                    isUpdateSession = true;
                }

                var isCheck = !string.IsNullOrEmpty(sessionSecretCode) &&
sessionSecretCode.Equals(answer.SecretCheckValue);

                if (!isCheck.Equals(answer.BoolValue))
                {
                    ModelState.AddModelError($@"Answers[{answer.ArrayId}].SecretValueError",
ErrorMessage.ServerError.GetValue(CultureInfo.CurrentUICulture.Name)?.ToString() ?? String.Empty);
                }

                _userSettingsSource.ClearSecretCode(request.AnswerProcedureId, answer.QuestionId);
            }

            if (ModelState.IsValid)
            {
                //TODO: saveAnswersResult обработать
                var saveAnswersResult = await
_verificationSource.SaveAnswersAsync(request.AnswerProcedureId, request.ScreenNumber, request.Answers);
            }
            return await PartialLoad(request.DivName, request.RecordId, request.PageName,
isUpdateSession);
        }

        #region Private
        private async Task<VerificationScreenViewModel> GetVerificationScreenViewModel(string pageName,
string divName, Guid recordId, bool updateSession)
        {
            var (body, profile) = await _pageTemplateSource.GetVerificationAsync(pageName, divName);

            var resultQuestions = await _verificationSource.GetQuestionsAsync(profile.QuestionName,
recordId, null);
            var result = new VerificationScreenViewModel
            {
                VerificationScreen = resultQuestions,
                RecordId = recordId,
                ProcedureName = profile.QuestionName,
                DivName = divName,
                PhotoHeight = profile.PhotoHeight ?? DefaultPhotoHeight,
                PageName = pageName
            };

            if (!updateSession) return result;

            var secretItems =
                result.VerificationScreen.Questions.Where(s => s.FieldType ==
QuestionFieldType.Secret);
            foreach (var secretItem in secretItems)
            {
                _userSettingsSource.SetSecretCode(result.VerificationScreen.AnswerProcedureId,
secretItem.Id,
                    secretItem.SecretValue);
            }

            _logger.LogInformation(@"Get verification questions={verificationName} for
record={recordId}, answer procedureId={answerProcedureId}
                , screen number={screen}, state={state}"
                , result.VerificationScreen.Procedure.SystemName, recordId
                , result.VerificationScreen.AnswerProcedureId, result.VerificationScreen.Screen.Number
                , result.VerificationScreen.State);

            return result;
        }
        #endregion
    }
}
```

Administrative Panel

Фрагмент модуля редактирования пользователей:

Identityserver/DS.IdentityServer.AdminPanel.Services/APServices/Users/UserService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using DS.IdentityServer.AdminPanel.Services.Exceptions;
using DS.IdentityServer.AdminPanel.Services.Interfaces;
using DS.IdentityServer.AdminPanel.Services.Models;
using DS.IdentityServer.AdminPanel.Services.Utilities;
using DS.IdentityServer.DAL.Enums;
using DS.IdentityServer.DAL.Model;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DS.IdentityServer.AdminPanel.Services.APServices.Users
{
    public class UserService : AbstractService, IUserService
    {
        private readonly ISecurityService _security;

        public UserService(
            ISEntities context,
            ICurrentLanguageAccessor language,
            ILogger<UserService> logger,
            IMapper mapper,
            ISecurityService security) : base(context, language, logger, mapper)
        {
            _security = security;
        }

        #region Users CRUD

        public Task<List<UserTableInfoModel>> GetListAsync()
        {
            var query = GetUserTempModels(_context.Users);

            return query.ProjectTo<UserTableInfoModel>(_mapperConfig).ToListAsync();
        }

        public async Task<(List<UserTableInfoModel>, TablePagingModel)> GetListAsync(TablePagingModel
paging)
        {
            IQueryable<DAL.Model.Users> query = _context.Users;
            if (!string.IsNullOrWhiteSpace(paging.FilterQuery))
            {
                query = query.Where(a => a.Name.ToLower().Contains(paging.FilterQuery.ToLower())
                                        || a.Login.ToLower().Contains(paging.FilterQuery.ToLower()));
            }
            var (pagingQuery, newPaging) = await QueryExtension.GetPagingListAsync(query, paging);
            var tempUserQuery = GetUserTempModels(pagingQuery);

            var userList = await
tempUserQuery.ProjectTo<UserTableInfoModel>(_mapperConfig).ToListAsync();
            return (userList, newPaging);
        }

        protected virtual IQueryable<GetUserTempModel> GetUserTempModels(IQueryable<DAL.Model.Users>
query)
        {
            var q = (from user in query.AsNoTracking()
                join lt in
_context.UsersLoginTypes.SelectDictionary(_language.CurrentLanguage).AsNoTracking()
                    on user.LoginTypeId equals lt.Id
                select new GetUserTempModel() {User = user, LoginType = lt});
            return q;
        }

        /// <inheritdoc/>
        public async Task<UserInfoModel> GetAsync(Guid id)
        {
            UserInfoModel userModel;
            if (id == Guid.Empty || (userModel = await
_context.Users.ProjectTo<UserInfoModel>(_mapperConfig).FirstOrDefaultAsync(u => u.Id == id)) == null)
            {
                throw new EditUserException("This User doesn't exist");
            }

            return userModel;
        }

        public async Task<UserInfoModel> UpdateAsync(UserInfoModel model)
        {
            DAL.Model.Users user;
            if (model.Id == Guid.Empty || (user = await _context.Users.FirstOrDefaultAsync(u => u.Id ==
model.Id)) == null)
            {
                throw new EditUserException("This User doesn't exist");
            }

            var oldModel = _mapper.Map<DAL.Model.Users, UserInfoModel>(user);

            _mapper.Map(model, user);

            await _context.SaveChangesAsync();

            var newModel = _mapper.Map<DAL.Model.Users, UserInfoModel>(user);
            _logger?.LogInformation("{object} ({id}) has been updated. Old: {@old} New: {@new}",
                "User", user.Id, oldModel, newModel);

            return newModel;
        }

        public async Task<UserInfoModel> CreateAsync(UserInfoModel model)
        {
            var user = _mapper.Map<UserInfoModel, DAL.Model.Users>(model);

            user.LoginTypeId = model.LoginTypeId;

            if (user.LoginTypeId == (short)LoginType.Local)
            {
                user.PasswordSalt = _security.GetSalt(64);
                user.PasswordHash = _security.HashPassword(user.PasswordSalt, model.Password);
            }

            await _context.Users.AddAsync(user);
            await _context.SaveChangesAsync();

            var newModel = _mapper.Map<DAL.Model.Users, UserInfoModel>(user);
            _logger?.LogInformation("{object} ({id}) has been created. User: {@new}",
                "User", user.Id, newModel);

            return newModel;
        }

        public async Task DeleteAsync(Guid id)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u => u.Id == id);
            var oldModel = _mapper.Map<DAL.Model.Users, UserInfoModel>(user);

            _context.Remove(user);
            await _context.SaveChangesAsync();

            _logger?.LogInformation("{object} ({id}) has been deleted. User: {@old}",
                "User", user.Id, oldModel);
        }
```

Authentication Service

Модуль доступа к пользователям:

```
Identityserver/DS.IdentityServer.BL/Managers/UsersStore.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace DS.IdentityServer.BL.Managers
{
    using DAL;
    using DAL.Model;

    public class UsersStore : IUsersStore
    {
        private readonly ISEntities _entities;
        private readonly IPasswordValidator _pswValidator;

        public UsersStore(ISEntities entities, IPasswordValidator pswValidator) {
            _entities = entities ?? throw new ArgumentNullException(nameof(entities));
            _pswValidator = pswValidator ?? throw new ArgumentNullException(nameof(pswValidator));
        }

        public async Task<(ValidateResult, IUser?)> ValidateCredentials(
            string login, string? password, string? client) {
            //seek user with that login
            Users user = await _entities.Users.Where(i => i.Login ==
login).AsNoTracking().FirstOrDefaultAsync();
            if (user == null)
                return (ValidateResult.InvalidLoginOrPassword, null);

            //validate password
            if (!_pswValidator.Validate(user, password))
                return (ValidateResult.InvalidLoginOrPassword, null);

            //check user can login to client
            DAL.Enums.CheckUserClientResult result = await _entities.CheckUserClient(user.Id, client);
            return result switch
            {
                DAL.Enums.CheckUserClientResult.Allowed => (ValidateResult.Success, user),
                DAL.Enums.CheckUserClientResult.ClientNotAllowed => (ValidateResult.ClientNotAllowed,
null),
                DAL.Enums.CheckUserClientResult.Inactive => (ValidateResult.InactiveUser, null),
                _ => (ValidateResult.UnknownReason, null),
            };
        }

        /// <summary>
        /// Can user login to specific client or not
        /// </summary>
        /// <param name="userId">User's id</param>
        /// <param name="client">Client which client want to login</param>
        /// <returns>true if can login</returns>
        public async Task<bool> CanLogin(Guid userId, string? client) {
            //check user can login to client
            DAL.Enums.CheckUserClientResult result = await _entities.CheckUserClient(userId, client);
            return result == DAL.Enums.CheckUserClientResult.Allowed;
        }

        public async Task<IUser?> Search(Guid id) {
            return await _entities.Users.Where(i => i.Id == id).AsNoTracking().FirstOrDefaultAsync();
        }

        public async Task<IEnumerable<Guid>> GetUserProjectsIds(Guid userId, string client) {
            var list = await _entities.VUsersClients
                .Where(i => i.UserId == userId && i.Client.SystemName == client && i.ProjectId != null)
                .Select(i => i.ProjectId)
                .ToArrayAsync();
            #pragma warning disable CS8629 // Nullable value type may be null.
            return list.Select(i => i.Value);
            #pragma warning restore CS8629 // Nullable value type may be null.
        }

        /// <summary>
        /// Returns list of available clients for user
        /// </summary>
        /// <param name="userId"></param>
        /// <returns></returns>
        public async Task<IEnumerable<string>> GetUserClients(Guid userId) {
            return  await _entities.VUsersClients
                .Where(i => i.UserId == userId)
                .Select(i => i.Client.SystemName)
                .Distinct() //TODO: is need here?
                .ToArrayAsync();
        }

        /// <summary>
        /// Returns list of roles for user and client
        /// </summary>
        /// <param name="userId"></param>
        /// <param name="client"></param>
        /// <returns></returns>
        public async Task<IEnumerable<string>> GetRoles(Guid userId, string client, Guid? projectId) {
            var q = _entities.VUsersRoles
                .Where(i => i.UserId == userId && i.Role.Client.SystemName == client);
            q = projectId.HasValue
                ? q.Where(i => i.ProjectId == null || i.ProjectId == projectId.Value)
                : q.Where(i => i.ProjectId == null);

            return await q
                .Select(i => i.Role.SystemName)
                .Distinct() //TODO: is need here?
                .ToArrayAsync();
        }
    }
}
```

Process Management Service

Модуль запуска процессов

```
Processengine/PE.Executor.Core/Queue/DbProcessStarter.cs

using PE.DAL;
using System;
using System.Data.Common;

using PE.Executor.DAL;
using Newtonsoft.Json.Linq;
using System.Threading.Tasks;
using PE.Executor.Core.Abstractions;
using PE.Executor.DAL.DataTypes;
using Npgsql;
using PE.Executor.Exceptions;
using PE.Executor.DataContract;

namespace PE.Executor.Core.Queue
{
    public class DbProcessStarter : IDbProcessStarter
    {
        private readonly IProcessQueries _queries;

        public DbProcessStarter(IProcessQueries queries)
        {
            _queries = queries ?? throw new ArgumentNullException(nameof(queries));
        }

        public async Task<Guid?> CreateProcessAsync(DbConnection conn, DbTransaction trans,
            string token, string action, JRaw? inData, Guid externalId)
        {
            try {
                //search process by token and action
                SearchProcessResult? searchResult = await _queries.SearchProcessAsync(conn, trans,
token, action);
                if (searchResult == null)
                    return null;

                //create new instance of process
                Guid id = await _queries.CreateProcessAsync(conn, trans, searchResult.FrontId,
searchResult.ProcessId,
                    inData?.ToString(), externalId);

                //returns created process id
                return id;
            } catch (NpgsqlException e) {
                if (e.SqlState == PostgresErrorCodes.DataException)
                    throw new CreateProcessException(action, token, externalId, e);
                else
                    throw;
            }
        }

        public async Task<StartProcessState> StartProcessAsync(DbConnection conn, DbTransaction trans,
Guid processInstanceId)
        {
            try {
                var state = await _queries.StartProcessAsync(conn, trans, processInstanceId);
                await _queries.QueueSeedAsync(conn, trans, processInstanceId);
                return state;
            } catch (NpgsqlException e) {
                if (e.SqlState == PostgresErrorCodes.DataException)
                    throw new StartProcessException(processInstanceId, e);
                else
                    throw;
            }
        }

        /// <inheritdoc/>
        public async Task<StartProcessResult> CreateSubprocessAsync(
            DbConnection conn,
            DbTransaction trans,
            IStepInfo step,
            JToken inData)
        {
            if (step.SubprocessTemplateMainId == null)
                throw new NullReferenceException($"Can't start subprocess because 
{nameof(step.SubprocessTemplateMainId)} is null");

            //search process template
            Guid tmplId = await _queries.SearchProcessAsync(conn, trans, 
step.SubprocessTemplateMainId.Value)
                ?? throw new InvalidOperationException($"Can't find active process for 
process_template_main_id {step.SubprocessTemplateMainId.Value}");

            //serialize indata
            string? inDataRaw = null;
            if (inData != null && inData.Type != JTokenType.Null)
                inDataRaw = inData.ToString();

            //create instance of process
            Guid instId = await _queries.CreateSubProcessAsync(conn, trans, tmplId, step.StepId, 
inDataRaw)
                ?? throw new InvalidOperationException($"Create process returns null for process id 
{tmplId}");

            //start process
            var state = await _queries.StartProcessAsync(conn, trans, instId);
            if (state == StartProcessState.Started)
                await _queries.QueueSeedAsync(conn, trans, instId);

            return new StartProcessResult(instId, state);
        }

        /// <inheritdoc/>
        public Task<ProcessResult?> QueryProcessResultAsync(
            DbConnection conn,
            DbTransaction trans,
            Guid instProcessId)
        {
            return _queries.SelectProcessResultAsync(conn, trans, instProcessId);
        }
    }
}
```

Interest Calculation Service
Фрагмент функций в базе данных

```
create function charges._uni_get_daily_percents(
    _product_version_id charges.products_info.product_version_id%type,
    _xdate date
)
    returns table (
        percent_day_total numeric(9, 4),
        details jsonb
    )
    language plpgsql volatile as $$
declare
    _charge_info record;
    _percent_day numeric(9, 4);
    _percent_day_total numeric(9, 4);
    _details_categories_ids int[];
    _details_percents numeric[];
    _details jsonb;
    _i int;
begin
    _percent_day_total = 0.0;

    for _charge_info in (
        select *
        from charges.v_charges_info ci
        where ci.product_version_id = _product_version_id
            and ci.is_interest -- Interest
    ) loop
        select percent into _percent_day from charges._get_percents(_charge_info,
_xdate); -- Possible percent_year->percent(_day) conversion

        _details_categories_ids = array_append(_details_categories_ids,
_charge_info.category_id);
        _details_percents = array_append(_details_percents, _percent_day);
        _percent_day_total = _percent_day_total + _percent_day;
    end loop;

    _details = '[]'::jsonb;
    for _i in 1..array_length(_details_categories_ids, 1) loop
        _details = _details || jsonb_build_array(
            jsonb_build_object(
                'category_id', _details_categories_ids[_i],
                'percent_day', _details_percents[_i],
                'weight', round(_details_percents[_i] / _percent_day_total, 6)
            )
        );
    end loop;


    -- ---

    return query
        select _percent_day_total, _details
    ;
end
```

```
$$;


create function charges.uni_create_schedule(
    _agreement_id agreements.agreements.id%type,
    _charges_from agreements.agreements.charges_from%type default null
)
    returns charges.schedules.id%type
    language plpgsql volatile as $$
declare
    _agreement_row agreements.agreements%rowtype;
    _agreement_jsonb jsonb;
    _schedule_id charges.schedules.id%type;
    _prev_schedule_tr_date charges.schedules_periods.tr_date%type;
    _schedule_period_id charges.schedules_periods.id%type;
    _row record;
    _cnt integer;
    _schedules_ids uuid[];
begin
    -- Lock agreement (at row level)
    select *
        into _agreement_row
    from agreements.agreements
    where id = _agreement_id
    for update;

    if _charges_from is null then
        _charges_from = _agreement_row.charges_from;
    end if;

    -- Guardian: no existing schedule

    if _charges_from is null then
        raise 'Unable to detect charges_from' using errcode = 'data_exception';
    elsif _agreement_row.charges_from is not null and _agreement_row.charges_from
!= _charges_from then
        raise 'charges_from values confilct' using errcode = 'data_exception';
    end if;

    select count(*), array_agg(id)
        into _cnt, _schedules_ids
    from charges.schedules
    where agreement_id = _agreement_id
    ;
    if _cnt > 1 then
        raise 'Too many schedules exists' using errcode = 'data_exception';
    elsif _cnt = 1 then
        return _schedules_ids[1]; -- return existing schedule ID
    else
        -- Need to create schedule
    end if;

    -- ---

    insert into charges.schedules (
        agreement_id
    ) values (
        _agreement_id
    ) returning id into _schedule_id;

    _agreement_jsonb = to_jsonb(_agreement_row);
    _prev_schedule_tr_date = null;

    for _row in (
        select *
        from charges._uni_get_schedule(_agreement_jsonb, _charges_from)
    ) loop
        if _prev_schedule_tr_date is null or _prev_schedule_tr_date !=
_row.tr_date then
            _prev_schedule_tr_date = _row.tr_date;

            insert into charges.schedules_periods (
                schedule_id,
                tr_date,
                principal_amount_rest, period_length
            ) values (
                _schedule_id,
                _row.tr_date,
                _row.principal_amount_rest, _row.period_length
            ) returning id into _schedule_period_id;
        end if;

        insert into charges.schedules_parts (
            schedule_period_id, category_id, amount, amounts_everyday
        ) values (
            _schedule_period_id, _row.category_id, _row.amount,
_row.amounts_everyday
        );
    end loop;

    update agreements.agreements
    set charges_from = _charges_from
    where id = _agreement_id
    ;

    return _schedule_id;
end
$$;
```

Product Service

Модуль контроллер продуктов

```
Productsengine/DS.ProductEngine.Web/Controllers/ProductController.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DS.ProductEngine.Models.Enums;
using DS.ProductEngine.Models.Model;
using DS.ProductEngine.Service.Extensions;
using DS.ProductEngine.Service.Interfaces;
using DS.ProductEngine.Web.Extensions;
using DS.ProductEngine.Web.Factory;
using DS.ProductEngine.Web.IdentityServerModel;
using DS.ProductEngine.Web.Models;
using DS.ProductEngine.Web.Models.ChargesCategories;
using DS.ProductEngine.Web.Models.Product;
using DS.ProductEngine.Web.Services.Abstractions;
using DS.ProductEngine.Web.Services.Services.Params.Models;
using DS.ProductEngine.Web.Services.Validators;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DS.ProductEngine.Web.Controllers
{
    public class ProductController : Controller
    {
        private readonly ICRUDService<Products> _productsService;
        private readonly ICRUDService<ProductsInfo> _productInfoService;
        private readonly IProductFactory _productFactory;
        private readonly IDictionaryService _dictionaryService;
        private readonly ILoanValidator _loanValidator;
        private readonly IParamsService _paramsService;
        private readonly IProductInfoService _piService;
        private readonly PEEntities _context;
        private readonly IChargesInfoService _chargesInfoService;
        private readonly IChargesInfoDaysService _chargesInfoDaysService;
        private readonly IProductVersionServices _productVersionServices;

        private static ProductCreate _productCreate;

        public ProductController(ICRUDService<Products> productsService,
            ICRUDService<ProductsInfo> productInfoService,
            IChargesInfoService chargesInfoService,
            IProductVersionServices productVersionServices,
            IChargesInfoDaysService chargesInfoDaysService,
            IProductFactory productFactory,
            IDictionaryService dictionaryService,
            ILoanValidator loanValidator,
            IParamsService paramsService,
            IProductInfoService piService,
            PEEntities context)
        {
            _productsService = productsService;
            _productVersionServices = productVersionServices;
            _productInfoService = productInfoService;
            _chargesInfoService = chargesInfoService;
            _productFactory = productFactory;
            _dictionaryService = dictionaryService;
            _loanValidator = loanValidator;
            _paramsService = paramsService;
            _piService = piService;
            _context = context;
            _chargesInfoDaysService = chargesInfoDaysService;
        }

        [ISRightAuthorize(IdentityServerRights.ProductsRead)]
        public async Task<ActionResult> Index()
        {
            return View(await _productFactory.ProductList());
        }

        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<ActionResult> Create()
        {
            return View(await _productFactory.CreateProductFormLoad());
        }

        [HttpPost]
        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<CreateMainInfoModel> CreateMainInfo(ProductCreate model)
        {
            var responseModel = new CreateMainInfoModel();

            if (!ModelState.IsValid)
            {
                Response.StatusCode = 400;
                var modelErrors = ModelState.AllErrors();
                responseModel.Errors = modelErrors;
                return responseModel;
            }

            model.AlgorithmTypeId = (int)AlgorithmTypeEnum.UNI;

            return await _productFactory.CreateMainInfo(model);
        }

        [HttpPost]
        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<CreateProductInformationModel> CreateProductInformation(Guid productId,
[FromForm]ProductsInfoCreate model, Guid? productInfoId)
        {
            var response = new CreateProductInformationModel();

            if (model.ProductTypeId == 2 && model.PeriodUnitId == null && model.PeriodLength == null)
            {
                ModelState.AddModelError("PeriodUnitId", ErrorMessages.RequiredField);
                ModelState.AddModelError("PeriodLength", ErrorMessages.RequiredField);
            }

            if ((model.PeriodUnitId != null && model.PeriodUnitId != 0) && model.PeriodLength == null)
            {
                ModelState.AddModelError("PeriodLength", ErrorMessages.RequiredField);
            }

            if ((model.AnnuityRoundMethodId != null && model.AnnuityRoundMethodId != 0) &&
model.AnnuityRoundPower == null)
            {
                ModelState.AddModelError("AnnuityRoundPower", ErrorMessages.RequiredField);
            }

            var (isValid, errors) = _loanValidator.Validate(model);
            if (!isValid)
            {
                foreach (var (fieldName, message) in errors)
                {
                    ModelState.AddModelError(fieldName, message);
                }
            }

            if (!ModelState.IsValid)
            {
                Response.StatusCode = 400;
                var modelErrors = ModelState.AllErrors();
                response.Errors = modelErrors;
                return response;
            }

            model.Id = productInfoId.GetValueOrDefault();
            if (model.Id == Guid.Empty)
            {
                var productInfo = await _piService.CreateAsync(productId, model);

                response.ProductVersionId = productInfo.ProductVersionId;
                response.ProductInformationId = productInfo.Id;
                response.ProductVersion = productInfo.ProductVersion.Version;
            }
            else
            {
                var productInfo = await _piService.UpdateAsync(model.Id, model);

                var lastVersion = await _productVersionServices.GetLastVersionAsync(productId);

                response.ProductVersionId = productInfo.ProductVersionId;
                response.ProductInformationId = productInfo.Id;
                response.ProductVersion = lastVersion.Version;
            }

            //await SaveProductInfo(productId, model, response);

            return response;
        }

        private async Task SaveProductInfo(Guid productId, ProductsInfoCreate model,
CreateProductInformationModel response)
        {
            var productInfo = await _productFactory.ProductInfoFormToProductInfoModel(model);

            // перезаписываются и при создании и при редактировании
            productInfo.IsUnrestrictedEarlyRepayment = true;
            productInfo.FirstDayReturnActionId = 0;
            productInfo.EarlyRepaymentModeId = 0;
            productInfo.IsFirstDayReturnAllowed = true;

            if (model.Id == Guid.Empty)
            {
                productInfo.ProductVersion = await _productVersionServices.CreateNewVersion(productId,
productInfo);

                var result = await _productInfoService.Insert(productInfo);

                response.ProductVersionId = result.ProductVersionId;
                response.ProductInformationId = result.Id;
                response.ProductVersion = productInfo.ProductVersion.Version;
            }
            else
            {
                await _productInfoService.Update(model.Id, productInfo);

                var lastVersion = await _productVersionServices.GetLastVersionAsync(productId);

                response.ProductVersionId = lastVersion.Id;
                response.ProductInformationId = lastVersion.ProductsInfo.Id;
                response.ProductVersion = lastVersion.Version;
            }
        }

        [HttpPost]
        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<ActionResult> ChargesInfoCreate(Guid productInfoId, Guid? id,
[FromForm]ChargesInfoEdit model)
        {
            try
            {
                if (model.CalcBaseId == null && model.CalcTypeId != 1)
                {
                    ModelState.AddModelError("CalcBaseId", ErrorMessages.RequiredField);
                }

                if ((model.PeriodTypeId != 2 && model.PeriodTypeId != 3) && model.CalcTypeId == 3)
                {
                    ModelState.AddModelError("CalcTypeId", "Поле заполняется только для
daily/daylist");
                }

                // https://metanit.com/sharp/aspnet5/19.6.php
                if (!ModelState.IsValid)
                {
                    Response.StatusCode = 400;
                    var modelErrors = ModelState.AllErrors();
                    return Json(modelErrors);
                }



                if (model.Id == Guid.Empty || model.Id == null)
                {
                    var newProductInfo = await
_productVersionServices.CreateNewVersionWithoutChange(productInfoId);
                    model.ProductInfoId = newProductInfo != Guid.Empty ? newProductInfo :
model.ProductInfoId;

                    var item = await _chargesInfoService.Insert(await
_productFactory.ChargesInfoEditFormToChargesInfoModel(model));

                    if (!string.IsNullOrEmpty(model.DaysList))
                    {
                        var daysListArray = model.DaysList.Split(",");
                        var daysList = daysListArray.Select(int.Parse).ToList();

                        if (daysList.Any())
                        {
                           await _chargesInfoDaysService.Insert(item.Id, daysList);
                        }
                    }
                }
                else
                {
                  await _chargesInfoService.Update(id.GetValueOrDefault(),
                        await _productFactory.ChargesInfoEditFormToChargesInfoModel(model));

                    if (!string.IsNullOrEmpty(model.DaysList))
                    {
                        var daysListArray = model.DaysList.Split(",");
                        var daysList = daysListArray.Select(int.Parse).ToList();

                        if (daysList.Any())
                        {
                            await _chargesInfoDaysService.Insert(id.GetValueOrDefault(), daysList);
                        }
                    }
                }

                var lastVersion = await _productVersionServices.GetLastVersionAsync(model.ProductId);

                var result = new CRUDResponseModel
                {
                    ProductInfoId = lastVersion.ProductsInfo.Id,
                    Version = lastVersion.Version,
                    ProductVersionId = lastVersion.Id
                };

                return Json(result);
            }
            catch (Exception err)
            {
                Response.StatusCode = 400;
                var resultError = new List<Error> {new Error("Error", $@"{err.Message}
{err.InnerException?.Message}")};
                return Json(resultError);
            }
```

```
        }

        [ISRightAuthorize(IdentityServerRights.ProductsRead)]
        public async Task<ActionResult> ChargesInfoList(Guid productInfoId)
        {
            var model = await _chargesInfoService.GetList(productInfoId);
            return PartialView("Partial/ChargesInfoList", await
_productFactory.ChargesInfoModelToChargesInfoForm(model));
        }

        [ISRightAuthorize(IdentityServerRights.ProductsRead)]
        public async Task<ActionResult> ChargesInfoListJson(Guid productInfoId)
        {
            // TODO: Do new Model special for List
            var model = await _chargesInfoService.GetList(productInfoId);
            return Json(await _productFactory.ChargesInfoModelToChargesInfoForm(model));
        }

        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<ActionResult> ChargesInfoItem(Guid chargesInfoId)
        {
            var model = await _chargesInfoService.GetItem(chargesInfoId);

            return PartialView("Partial/Create/ChargesInfoCreate", await
_productFactory.ChargeInfoModelToChargeInfoForm(model));
        }

        [ISRightAuthorize(IdentityServerRights.ProductsRead)]
        public async Task<ActionResult> ChargesInfoItemJson(Guid chargesInfoId)
        {
            var model = await _chargesInfoService.GetItem(chargesInfoId);
            var response = await _productFactory.ChargeInfoModelToChargeInfoForm(model);

            return Json(response);
        }

        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public async Task<ActionResult> ChargesInfoItemEdit(Guid chargesInfoId)
        {
            var model = await _chargesInfoService.GetItem(chargesInfoId);

            return PartialView("Partial/Edit/ChargesInfoEdit", await
_productFactory.ChargeInfoModelToChargeInfoForm(model));
        }

        [ISRightAuthorize(IdentityServerRights.ProductsRead)]
        public async Task<ActionResult> Edit(Guid id)
        {
            try
            {
                var model = await _productFactory.EditProductFormLoad(id);

                if (model.ProductsInfo != null)
                {
                    model.ProductsInfo.Id = model.ProductsInfo.ProductsInfoId;
                    model.ProductsInfo.ProductKindId = model.ProductKindId;
                }
                else
                {
                    model.ProductsInfo = new ProductsInfoCreate();
                    model.ChargesInfoCreate = new ChargesInfoCreate();
                    model.ChargesInfoEdit = new ChargesInfoEdit();
                }

                return View(model);
            }
            catch (Exception err)
            {
                Response.StatusCode = 404;
                return Ok(err.Message);
            }
        }

        [ISRightAuthorize(IdentityServerRights.ProductsReadWritePublish)]
        public async Task<ActionResult> Public(Guid id)
        {
            try
            {
                TempData["TempModel"] = "";
                await _productVersionServices.PublishVersion(id);
                return RedirectToAction(nameof(Index));
            }
            catch (ExceptionServiceExtension err)
            {
                if (err.Error.Code == 1)
                {
                    TempData["TempModel"] = "Продукт уже опубликован";
                }
                return RedirectToAction(nameof(Index));
            }
        }

        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public ActionResult Delete(Guid id)
        {
            TempData["TempModel"] = "НЕ РЕАЛИЗОВАНО";
            return RedirectToAction(nameof(Index));
        }

        [HttpPost]
        [ISRightAuthorize(IdentityServerRights.ProductsWrite)]
        public void ChargesInfoDeleteLocal(Guid chargesInfoId)
        {
            //site JS - удалить!!
            var item = _productCreate.ChargesInfoList.FirstOrDefault(s => s.Id == chargesInfoId);
            if (item != null) _productCreate.ChargesInfoList.Remove(item);
        }

        [HttpGet("[controller]/ProductsKinds")]
        [ISRightAuthorize(IdentityServerRights.DictionariesRead)]
        public async Task<IActionResult> GetProductsKinds()
        {
            var result = await _dictionaryService.GetProductsKindsAsync();

            return Json(result);
        }

        [HttpGet("[controller]/ProductsTypes")]
        [ISRightAuthorize(IdentityServerRights.DictionariesRead)]
        public async Task<IActionResult> GetProductsTypesItems()
        {
            var result = await _dictionaryService.GetProductsTypesAsync();

            return Json(result);
        }
    }
}
```

Restrictions Service

Модуль редактирования ограничений: Productslimits/DS.ProductsLimits.Service/PLServices/Limits/LimitService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using DS.ProductsLimits.DAL.EF.Model;
using DS.ProductsLimits.DAL.EF.Partial;
using DS.ProductsLimits.Service.Exceptions;
using DS.ProductsLimits.Service.Interfaces;
using DS.ProductsLimits.Service.PLServices.Limits.Models;
using DS.ProductsLimits.Service.PLServices.Projects.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DS.ProductsLimits.Service.PLServices.Limits
{
    public class LimitService : AbstractService, ILimitService
    {
        private readonly IFunctions _functions;

        public LimitService(
            ISEntities context,
            ICurrentLanguageAccessor language,
            ILogger<LimitService> logger,
            IMapper mapper,
            IFunctions functions) : base(context, language, logger, mapper)
        {
            _functions = functions;
        }

        public async Task<List<LimitsTypeModel>> GetLimitsTypesAsync()
        {
            var result = await _context.LimitsTypes
                .SelectDictionary(_language.CurrentLanguage)
                .ProjectTo<LimitsTypeModel>(_mapperConfig)
                .ToListAsync();

            return result;
        }

        public async Task<List<ActualLimitModel>> GetActualLimitsAsync(short limitTypeId, DateTime 
dateTime)
        {
            var funcView = _functions.SearchActualLimitsResult(_context, limitTypeId, dateTime);
            var tempQuery = GetActualLimitTempModel(funcView);
            var result = await tempQuery.ProjectTo<ActualLimitModel>(_mapperConfig).ToListAsync();

            return result;
        }

        protected virtual IQueryable<ActualLimitTempModel> GetActualLimitTempModel(
            IQueryable<SearchActualLimitsResult> query)
        {
            var q = from limit in query.AsNoTracking()
                join pt in 
_context.ProductsTypes.SelectDictionary(_language.CurrentLanguage).AsNoTracking()
                    on limit.ProductFeature.ProductTypeId equals pt.Id into grouping
                from p in grouping.DefaultIfEmpty()
                select new ActualLimitTempModel() {Limit = limit, ProductType = p};
            return q;
        }

        protected virtual IQueryable<ActualLimitTempModel> GetActualLimitTempModelOld(
            IQueryable<SearchActualLimitsResult> query)
        {
            var q = (from limit in query.AsNoTracking()
                join pt in
_context.ProductsTypes.SelectDictionary(_language.CurrentLanguage).AsNoTracking()
                    on limit.ProductFeature.ProductTypeId equals pt.Id
                select new ActualLimitTempModel() { Limit = limit, ProductType = pt });
            return q;
        }

        public async Task UpdateAsync(UpdateLimitModel model)
        {
            Limit limit;
            if (model.Id == 0 || (limit = await _context.Limits.FirstOrDefaultAsync(u => u.Id ==
model.Id)) == null)
            {
                throw new EditLimitException("This Limit doesn't exist");
            }

            var oldModel = _mapper.Map<Limit, UpdateLimitModel>(limit);

            _mapper.Map(model, limit);

            await _context.SaveChangesAsync();

            var newModel = _mapper.Map<Limit, UpdateLimitModel>(limit);

            _logger?.LogInformation("{object} ({id}) has been updated. Old: {@old} New: {@new}",
                "Limit", limit.Id, oldModel, newModel);
        }

        public async Task<CreateLimitModel> CreateAsync(CreateLimitModel model)
        {
            var productFuture = await _context.ProductsFeatures
                .FirstOrDefaultAsync(a => a.ProductTypeId == model.ProductTypeId &&
                                          a.MinAmount == model.MinAmount &&
                                          a.MaxAmount == model.MaxAmount &&
                                          a.MinTerm == model.MinTerm &&
                                          a.MaxTerm == model.MaxTerm);

            if (productFuture == null)
            {
                productFuture = new ProductsFeature()
                {
                    ProductTypeId = model.ProductTypeId,
                    MinAmount = model.MinAmount,
                    MaxAmount = model.MaxAmount,
                    MinTerm = model.MinTerm,
                    MaxTerm = model.MaxTerm,
                };

                await _context.ProductsFeatures.AddAsync(productFuture);
                await _context.SaveChangesAsync();

                _logger?.LogInformation("{object} ({id}) has been created. ProductsFeature: {@new}",
                    "ProductsFeature", productFuture.Id, productFuture);
            }

            var limit = new Limit()
            {
                DateFrom = model.DateFrom,
                LimitTypeId = model.LimitTypeId,
                ProjectId = model.ProjectId,
                Value = model.Value,
                ProductFeatureId = productFuture.Id,
            };

            await _context.Limits.AddAsync(limit);

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateException e)
            {
                throw new CreateLimitException("Creation Error");
            }

            model.Id = limit.Id;
            _logger?.LogInformation("{object} ({id}) has been created. Limit: {@new}",
                "Limit", limit.Id, model);

            return model;
        }
    }
}
```