Додаток А

Графічний матеріал атестаційної роботи

ГЮИК.509000.005

(позначення документу)

ЗАТВЕРДЖЕНО
ГЮИК.509000.005 – ЛУ

Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних

Графічний матеріал

ГЮИК.509000.005– ЛУ

ЛИСТІВ 12

2020 р.

Міністерство освіти і науки України
Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційної роботи
проф. Ситніков Д. Е.

Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних

Графічний матеріал

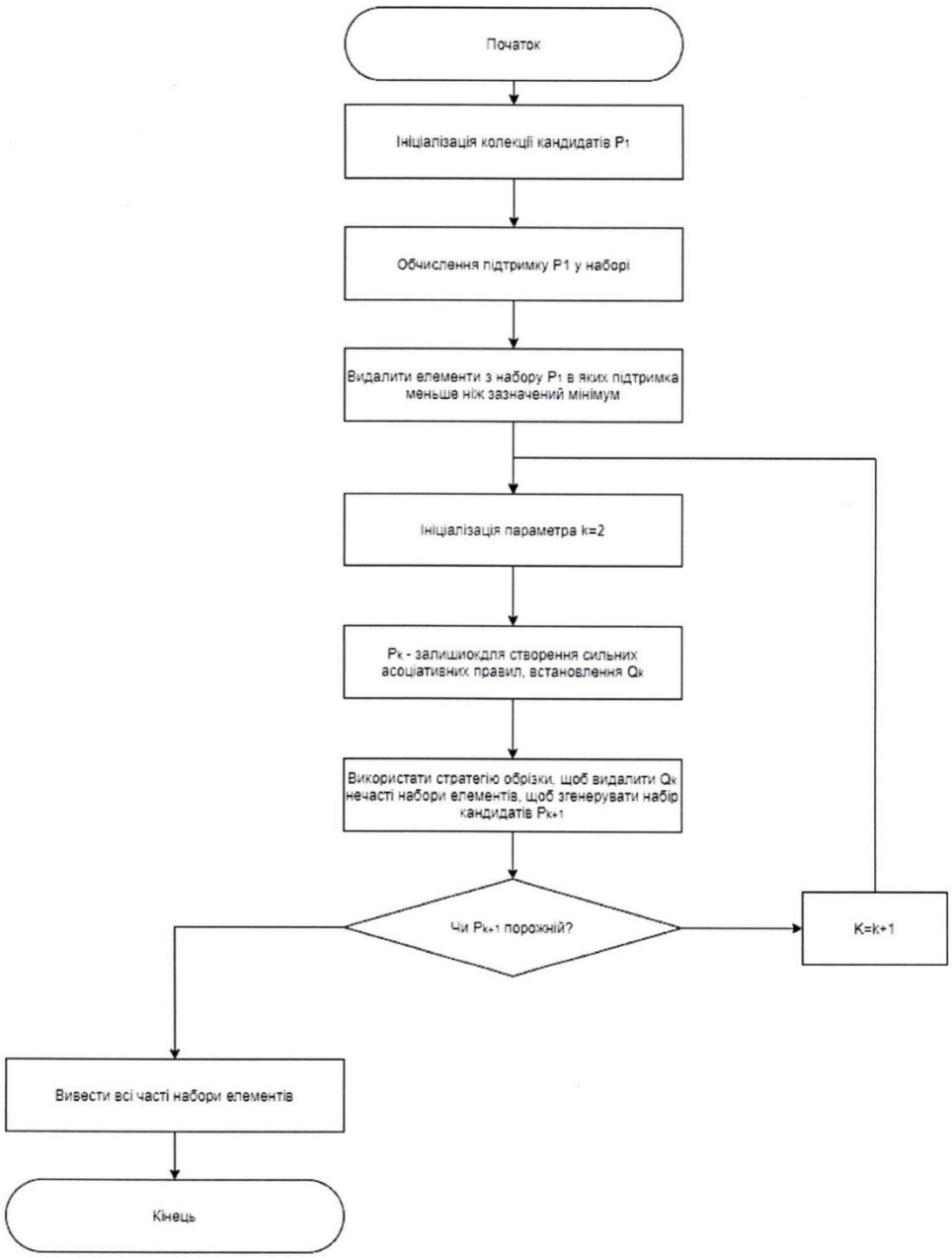
ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК. 509000.005 – ЛУ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Демченко О. Е.

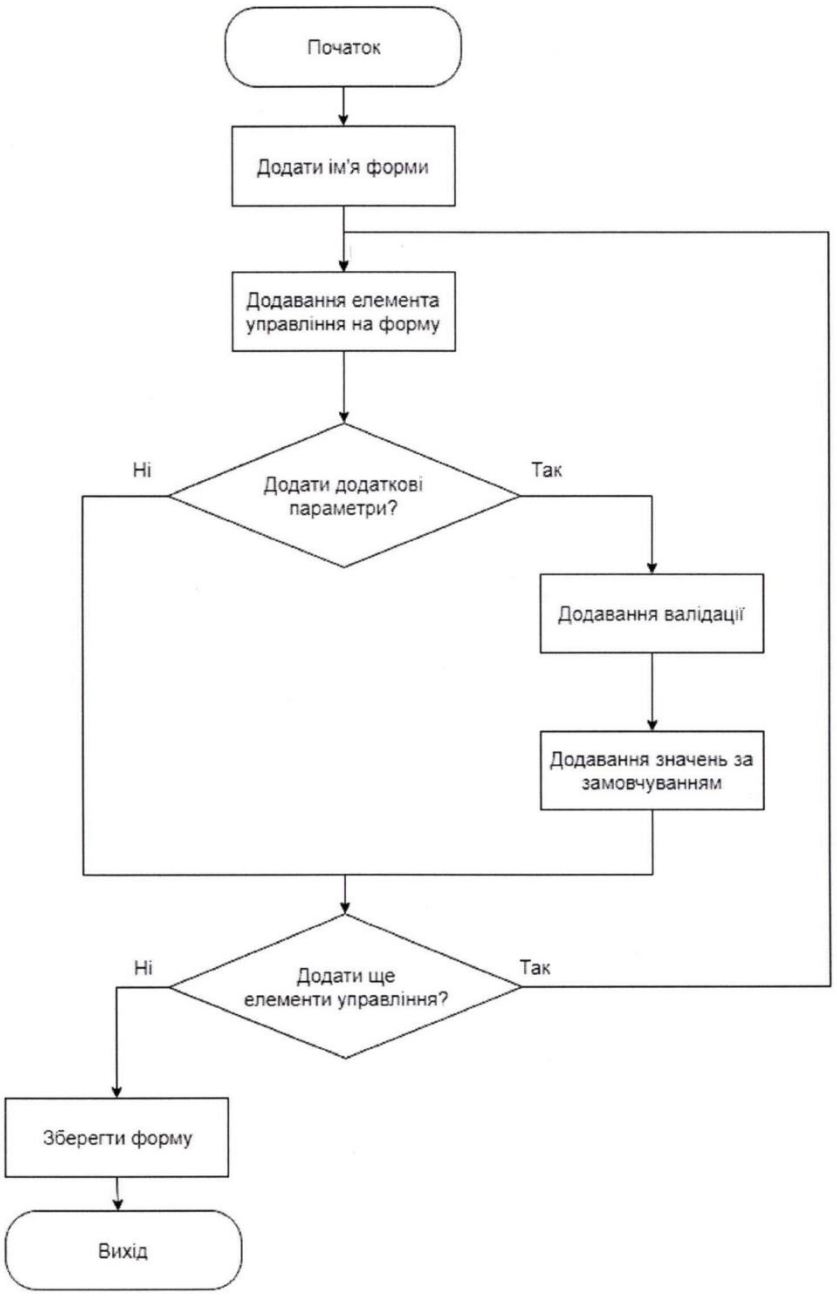
2020 р.

Алгоритм Apriori



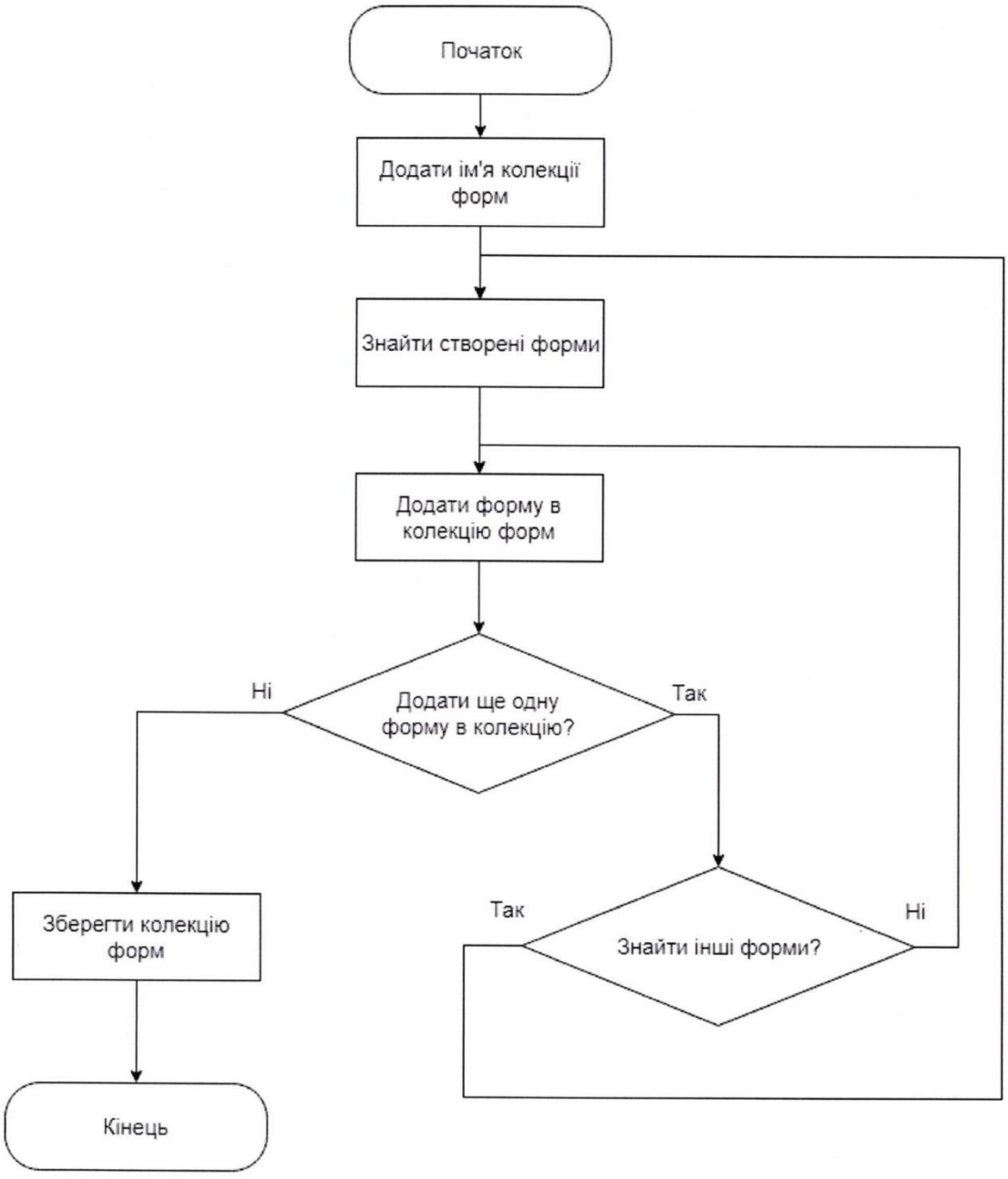
Розроб.	*Демченко О.Е.*			*Алгоритм Apriori*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 1*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Алгоритм створення колекції форм



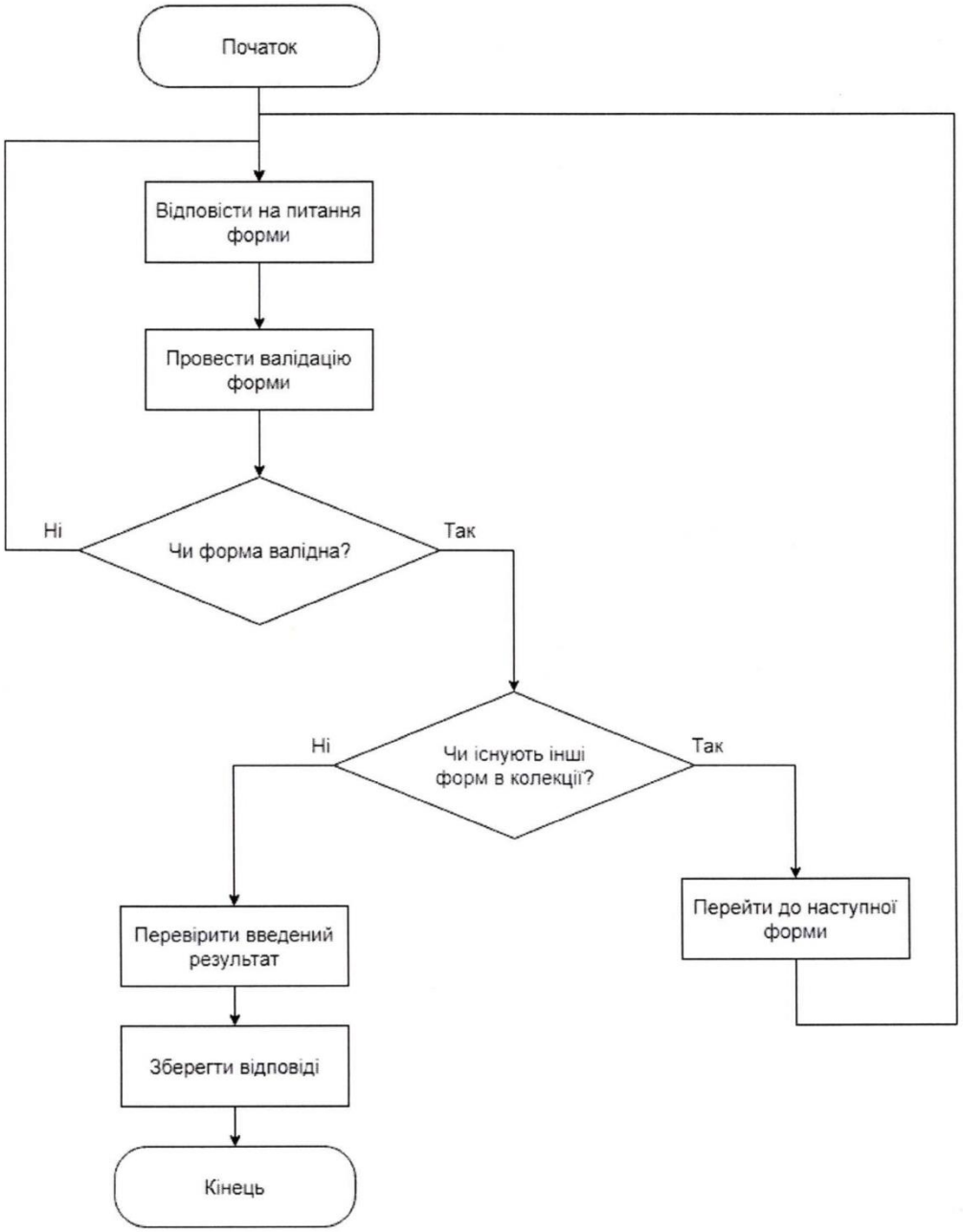
Розроб.	*Демченко О.Е.*			*Алгоритм створення форми*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 2*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Алгоритм створення колекції форм



Розроб.	*Демченко О.Е.*			*Алгоритм створення колекції форм*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 3*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Алгоритм відповіді на колекцію форм

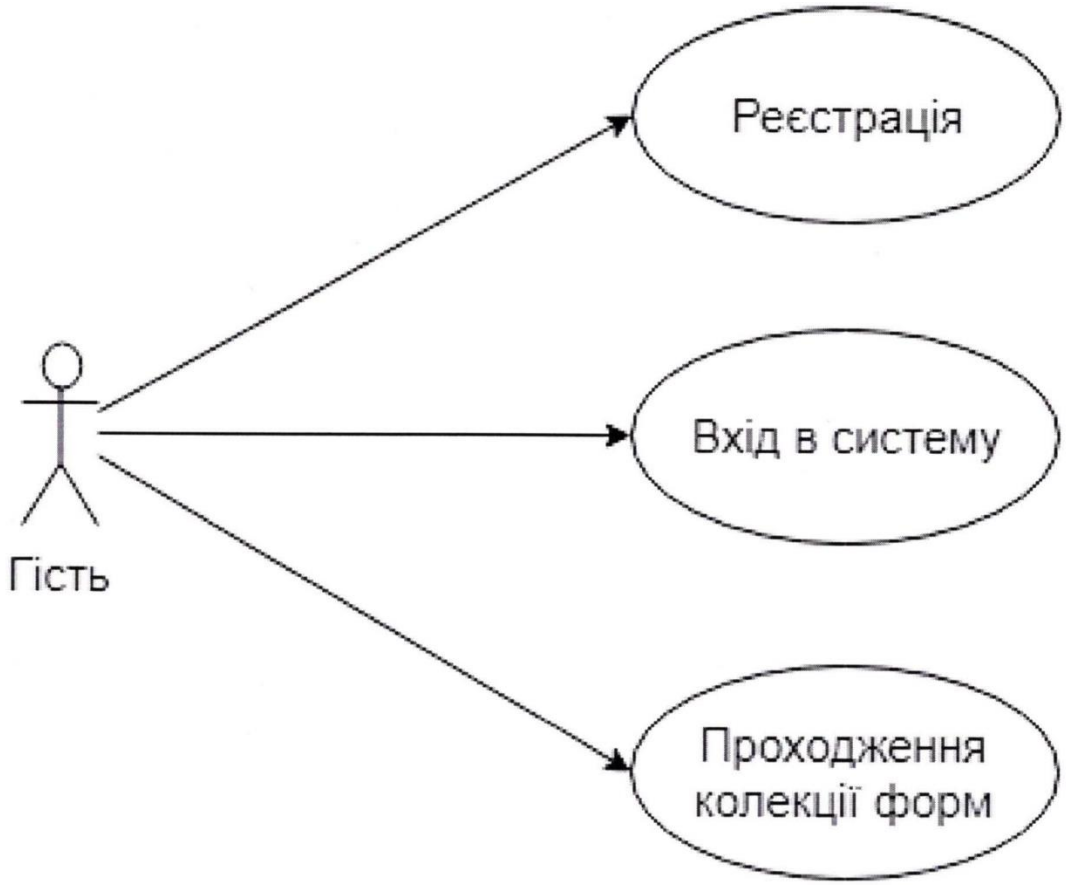


Розроб.	*Демченко О.Е.*			*Алгоритм відповіді на колекцію форм*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 4*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Алгоритм відповіді на колекцію форм

Розроб.	Демченко О.Е.			Алгоритм запуску аналізу колекції форм	
Перевір.	Ситніков Д. Е.				
Н. Контр.	Ситніков Д. Е.				
				СПРМ-18-2	Лист 5
Затверд.	Гребеннік І. В.			СТ	Листів 12

Діаграма прецедентів для користувача «Гість»



Розроб.	Демченко О.Е.			Діаграма прецедентів для користувача «Гість»	
Перевір.	Ситніков Д. Е.				
Н. Контр.	Ситніков Д. Е.				
				СПРМ-18-2	Лист 6
Затверд.	Гребеннік І. В.			СТ	Листів 12

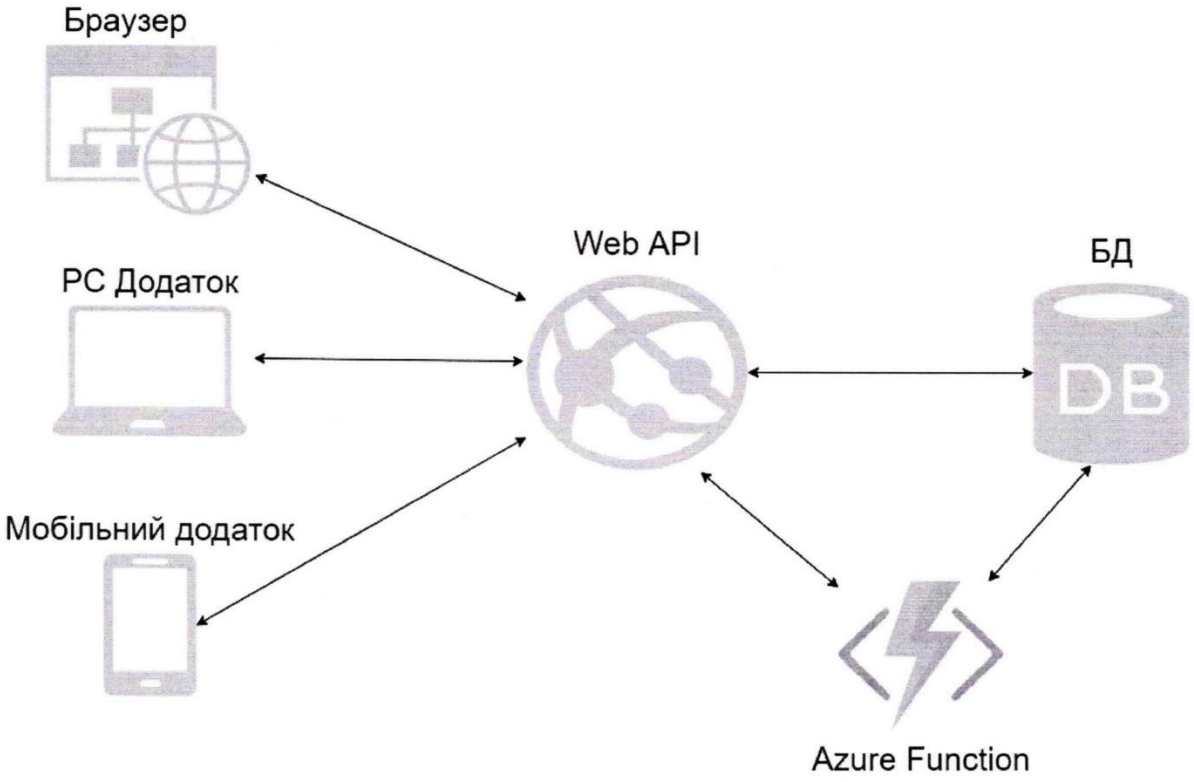
Діаграма прецедентів для користувача «Користувач»

Розроб.	*Демченко О.Е.*			*Діаграма прецедентів для користувача «Користувач»*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 7*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Діаграма прецедентів для користувача «Адміністратор»

Розроб.	*Демченко О.Е.*			*Діаграма прецедентів для користувача «Адміністратор»*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 8*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Узагальнена схема роботи Web API

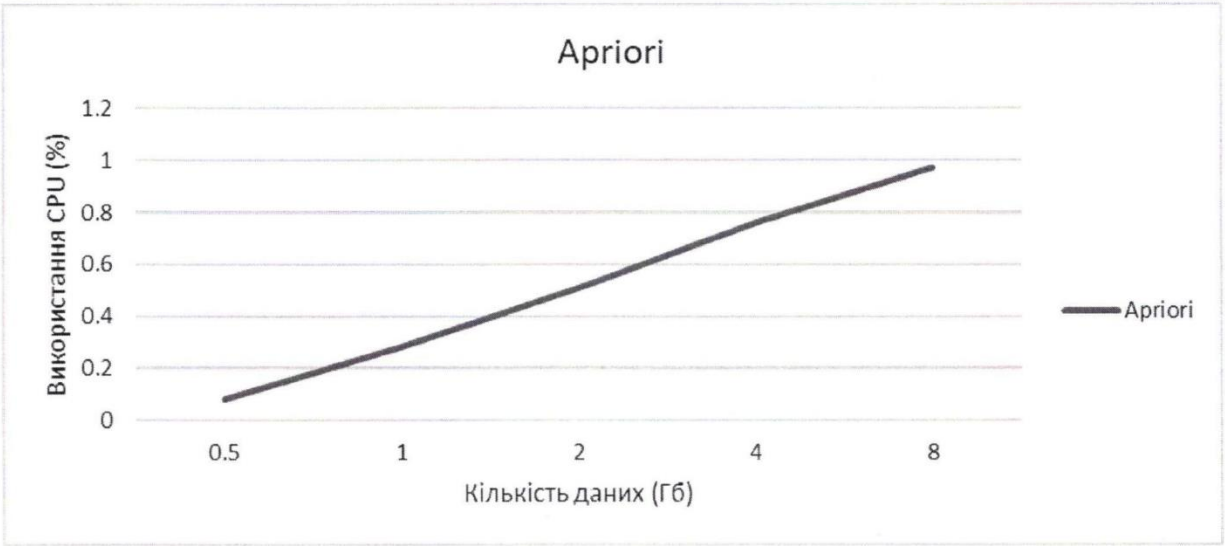


Розроб.	*Демченко О.Е.*			*Узагальнена схема роботи Web API*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРМ-18-2	*Лист 9*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Час прогону алгоритма Apriori

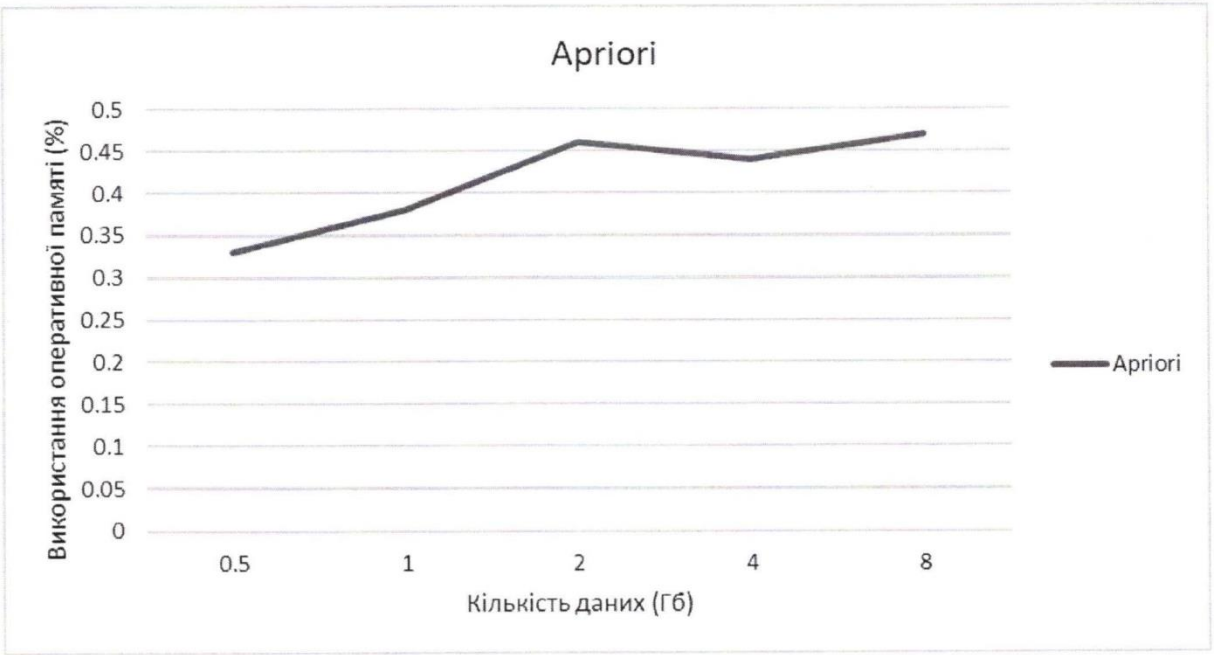
Розроб.	Демченко О.Е.			Час прогону алгоритма Apriori	
Перевір.	Ситніков Д. Е.				
Н. Контр.	Ситніков Д. Е.				
				СПРм-18-2	Лист 10
Затверд.	Гребеннік І. В.			СТ	Листів 12

Використання CPU алгоритмом Apriori



Розроб.	*Демченко О.Е.*			*Використання CPU алгоритмом Apriori*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРм-18-2	*Лист 11*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Використання оперативної памяті алгоритмом Apriori



Розроб.	*Демченко О.Е.*			*Використання оперативної памяті алгоритмом Apriori*	
Перевір.	*Ситніков Д. Е.*				
Н. Контр.	*Ситніков Д. Е.*				
				СПРм-18-2	*Лист 12*
Затверд.	*Гребеннік І. В.*			*СТ*	*Листів 12*

Додаток Б

Текст програми

ГЮИК.509000.005 – 01 12 01

(позначення документу)

ЗАТВЕРДЖЕНО
ГЮИК.509000.005 – 01 12 01 – ЛУ

Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних

Текст програми

ГЮИК.509000.005 – 01 12 01 – ЛУ

ЛИСТІВ 21

2020 р.

Міністерство освіти і науки України
Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційної роботи
проф. Ситніков Д. Е.

Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК.ГЮИК.509000.005 – 01 12 01 – ЛУ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Демченко О. Е.

2020 р.

BaseEntity

```
using System;

namespace FormsGenerator.Domain.Entities
{
   public class BaseEntity
   {
      public Guid Id { get; set; }
      public bool IsDeleted { get; set; }
   }
}
```

Control

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
   public abstract class Control : BaseEntity
   {
      public string Type { get; set; }
      public string Label { get; set; }
      public IEnumerable<ValidationRule> ValidationRules { get; set; }

      protected Control()
      {
         ValidationRules = Enumerable.Empty<ValidationRule>();
      }
   }
}
```

NameValuePair

```
namespace FormsGenerator.Domain.Entities
{
   public class NameValuePair
   {
      public string Name { get; set; }
      public string Value { get; set; }
      public bool IsChecked { get; set; }
   }
}
```

ControlSet

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
   public class ControlSet
   {
      public bool IsValid { get; set; }
      public IEnumerable<Control> Controls { get; set; }

      public ControlSet()
      {
         Controls = Enumerable.Empty<Control>();
      }
   }
}
```

TextBox

```
namespace FormsGenerator.Domain.Entities
{
   public class TextBox : Control
   {
      public string Value { get; set; }
   }
}
```

TextArea

```
namespace FormsGenerator.Domain.Entities
{
   public class TextArea : Control
   {
      public int Rows { get;set; }
      public string Value { get; set; }
   }
}
```

CheckBoxGroup

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
```

```
   public class CheckBoxGroup : Control
   {
      public IEnumerable<NameValuePair> Options { get; set; }
      public IEnumerable<string> Values { get; set; }

      public CheckBoxGroup()
      {
         Options = Enumerable.Empty<NameValuePair>();
         Values = Enumerable.Empty<string>();
      }
   }
}
```

DatePicker

```
using System;

namespace FormsGenerator.Domain.Entities
{
   public class DatePicker : Control
   {
      public DateTime? Value { get; set; }
   }
}
```

DropDownList

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
   public class DropDownList : Control
   {
      public bool IsMultiselect { get; set; }
      public IEnumerable<NameValuePair> Options { get; set; }
      public string Value { get; set; }

      public DropDownList()
      {
         Options = Enumerable.Empty<NameValuePair>();
      }
   }
}
```

RadioButtonGroup

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
   public class RadioButtonGroup : Control
   {
      public IEnumerable<NameValuePair> Options { get; set; }
      public string Value { get; set; }

      public RadioButtonGroup()
      {
         Options = Enumerable.Empty<NameValuePair>();
      }
   }
}
```

FormBase

```
using System;

namespace FormsGenerator.Domain.Entities
{
   public class FormBase : BaseEntity
   {
      public string Name { get; set; }
      public DateTime? LastModified { get; set; }
      public bool IsValid { get; set; }
      public FormType Type { get; protected set; }
   }
}
```

DynamicForm

```
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
   public class DynamicForm : FormBase
   {
      public bool IsMultiEntry { get; set; }
      public IEnumerable<ControlSet> ControlSets { get; set; }
      public IEnumerable<Control> Controls { get; set; }
```

```
        public DynamicForm()
        {
            ControlSets = Enumerable.Empty<ControlSet>();
            Controls = Enumerable.Empty<Control>();
            Type = FormType.Dynamic;
        }
    }
}
```

FormCollection

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace FormsGenerator.Domain.Entities
{
    public class FormCollection : BaseEntity
    {
        public string Name { get; set; }
        public DateTime LastModified { get; set; }
        public string SubmissionUrl { get; set; }
        public IEnumerable<Guid> Forms { get; set; }

        public FormCollection()
        {
            Forms = Enumerable.Empty<Guid>();
        }
    }
}
```

FormCollectionAnswer

```
using System;

namespace FormsGenerator.Domain.Entities.Forms
{
    public class FormCollectionAnswer : BaseEntity
    {
        public bool IsCompleted { get; set; }
        public DateTime SubmitDate { get; set; }
        public string UserAgent { get; set; }
        public Guid FormCollectionId { get; set; }
        public Guid? ClientId { get; set; }
        public FormCollectionSnapShot FormCollection { get; set; }
    }
```

}

FormService

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using FormsGenerator.Domain.Entities;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Filtering;
using FormsGenerator.Services.Contracts.Models.Filtering;
using FormsGenerator.Services.Contracts.Repositories;
using FormsGenerator.Services.PredicateBuilder;

namespace FormsGenerator.Services.DomainServices
{
   public class FormService : IFormService
   {
      private readonly IMapper _mapper;
      private readonly IRepository<FormBase> _formRepository;
      private readonly IRepository<DynamicForm> _dynamicFormRepository;
      private readonly IRepository<FormCollection> _formCollectionRepository;
      private readonly IFormFilterService _filterService;

      public FormService(
         IMapper mapper,
         IRepository<FormBase> formRepository,
         IRepository<FormCollection> formCollectionRepository,
         IFormFilterService filterService,
         IRepository<DynamicForm> dynamicFormRepository)
      {
         _mapper = mapper;
         _formRepository = formRepository;
         _formCollectionRepository = formCollectionRepository;
         _filterService = filterService;
         _dynamicFormRepository = dynamicFormRepository;
      }

      public async Task<FormBaseDto> GetById(Guid id)
      {
         return _mapper.Map<FormBaseDto>(
            await _formRepository.FindAsync(f => f.Id == id));
      }
```

```
public async Task<FormBaseDto> GetByName(string formName)
{
    return _mapper.Map<FormBaseDto>(
        await _formRepository.FindAsync(f => f.Name == formName));
}

public async Task<List<FormBaseDto>> GetAll()
{
    return _mapper.Map<List<FormBaseDto>>(await _formRepository.GetAllAsync());
}

public async Task<List<FormBaseDto>> GetAll(FormFilter filter)
{
    var predicate = _filterService.BuildFilterPredicate(filter);

    if (filter.InCollectionId != null)
    {
        var guids = await GetFromGuids(filter.InCollectionId);

        predicate = predicate.And(f => guids.Contains(f.Id));

        return _mapper.Map<List<FormBaseDto>>(await _formRepository.GetAllAsync(predicate));
    }

    return _mapper.Map<List<FormBaseDto>>(await _formRepository.GetAllAsync(predicate));
}

public Task<DynamicForm> Create(DynamicForm form)
{
    form.LastModified = DateTime.UtcNow;

    return _dynamicFormRepository.CreateAsync(form);
}

public Task<DynamicForm> Update(DynamicForm form)
{
    form.LastModified = DateTime.UtcNow;

    return _dynamicFormRepository.UpdateAsync(form);
}

public async Task Delete(Guid id)
{
    var form = await _formRepository.FindAsync(formBase => formBase.Id == id);
```

```
        if (form.Type == FormType.Static)
        {
            throw new InvalidOperationException($"Somebody has tried delete static Form thant has id{id.ToString()}");
        }

        await DeleteFromCollections(id);

        await _formRepository.RemoveAsync(id);
    }

    public async Task<bool> IsFormInUse(Guid id)
    {
        var collection = await _formCollectionRepository
            .FindAsync(coll => coll.Forms.Contains(id));

        return collection != null;
    }

    private async Task<List<Guid>> GetFromGuids(Guid? collectionId)
    {
        var collection = await _formCollectionRepository.FindAsync(c => c.Id == collectionId);

        return collection.Forms == null
            ? new List<Guid>()
            : collection.Forms.ToList();
    }

    private async Task DeleteFromCollections(Guid formId)
    {
        var collections = await _formCollectionRepository
            .GetAllAsync(collection => collection.Forms.Contains(formId));

        foreach (var formCollection in collections)
        {
            formCollection.Forms = formCollection.Forms.Where(fId => fId != formId);
            await _formCollectionRepository.UpdateAsync(formCollection);
        }
    }
  }
}
```

FormCollectionService

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;
using AutoMapper;
using FormsGenerator.Domain.Entities;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Filtering;
using FormsGenerator.Services.Contracts.Models.Filtering;
using FormsGenerator.Services.Contracts.Repositories;

namespace FormsGenerator.Services.DomainServices
{
   public class FormCollectionService : IFormCollectionService
   {
      private readonly IRepository<FormCollection> _formCollectionRepository;
      private readonly IRepository<FormBase> _formBase;
      private readonly IFormCollectionFilterService _filterService;
      private readonly IMapper _mapper;

      public FormCollectionService(
         IRepository<FormCollection> formCollectionRepository,
         IFormCollectionFilterService filterService,
         IMapper mapper,
         IRepository<FormBase> baseForm)
      {
         _formCollectionRepository = formCollectionRepository;
         _filterService = filterService;
         _mapper = mapper;
         _formBase = baseForm;
      }

      public async Task<FormCollectionDto> GetById(Guid id)
      {
         var formCollection = await _formCollectionRepository.FindAsync(fc => fc.Id == id);

         var formCollectionDto = _mapper.Map<FormCollectionDto>(formCollection);

         if (formCollectionDto != null)
         {
            formCollectionDto.Forms = formCollection.Forms != null
               ? await GetCollectionForms(formCollection.Forms)
               : new List<FormBaseDto>();
         }

         return formCollectionDto;
      }
```

```
public async Task<FormCollectionDto> GetByName(string formCollectionName)
{
    var formCollection = await _formCollectionRepository.FindAsync(fc => fc.Name == formCollectionName && !fc.IsDeleted);

    var formCollectionDto = _mapper.Map<FormCollectionDto>(formCollection);

    if (formCollectionDto != null)
    {
        formCollectionDto.Forms = formCollection.Forms != null
            ? await GetCollectionForms(formCollection.Forms)
            : new List<FormBaseDto>();
    }

    return formCollectionDto;
}

public async Task<List<FormCollectionDto>> GetAll()
{
    return _mapper.Map<List<FormCollectionDto>>(await _formCollectionRepository.GetAllAsync());
}

public async Task<List<FormCollectionDto>> GetAll(FormCollectionFilter filters)
{
    var predicate = _filterService.BuildFilterPredicate(filters);

    return _mapper.Map<List<FormCollectionDto>>(await _formCollectionRepository.GetAllAsync(predicate));
}

public Task Create(FormCollectionDto formCollectionDto)
{
    formCollectionDto.LastModified = DateTime.UtcNow;

    var formCollection = _mapper.Map<FormCollection>(formCollectionDto);

    return _formCollectionRepository.CreateAsync(formCollection);
}

public Task Update(FormCollectionDto formCollectionDto)
{
    formCollectionDto.LastModified = DateTime.UtcNow;

    var formCollection = _mapper.Map<FormCollection>(formCollectionDto);

    return _formCollectionRepository.UpdateAsync(formCollection);
}
```

```
        public Task Delete(Guid id)
        {
            return _formCollectionRepository.RemoveAsync(id);
        }

        private async Task<List<FormBaseDto>> GetCollectionForms(IEnumerable<Guid> formGuids)
        {
            if (formGuids == null) throw new ArgumentNullException(nameof(formGuids), $"{nameof(formGuids)} can't be null!");

            var returnForms = new List<FormBaseDto>();

            var forms = await _formBase.GetAllAsync(f => formGuids.Contains(f.Id));

            foreach (var formGuid in formGuids)
            {
                var form = forms.FirstOrDefault(f => f.Id == formGuid);

                returnForms.Add(_mapper.Map(form, form.GetType(), typeof(FormBaseDto)) as FormBaseDto);
            }

            return returnForms;
        }
    }
}
```

ExportService

```
using System.Collections.Generic;
using System.Linq;
using FormsGenerator.Domain.Entities;
using FormsGenerator.Services.Contracts.AdditionalServices.Export;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Models.Export;

namespace FormsGenerator.Services.DomainServices
{
    public class ExportService : IExportService
    {
        private readonly List<string> _ignorableControlTypes = new List<string> {"divider", "file"};
        private readonly IExportFileBuildersProvider _buildersProvider;

        public ExportService(IExportFileBuildersProvider buildersProvider)
        {
            _buildersProvider = buildersProvider;
        }
```

```
public List<FormCollectionSelectItemDto> CreateSelectOptions(
    ICollection<FormCollectionAnswerDto> formCollectionAnswers)
{
    var selectItems = new List<FormCollectionSelectItemDto>();
    foreach (var currentFormCollectionAnswerDto in formCollectionAnswers)
    {
        foreach (var currentFormBaseDto in currentFormCollectionAnswerDto.FormCollection.Forms)
        {
            AppendSelectItem(selectItems, currentFormBaseDto);
        }
    }

    return selectItems;
}

public string CreateAnswersFile(
    ICollection<FormCollectionSelectItemDto> selectedOptions,
    ICollection<FormCollectionAnswerDto> formCollectionAnswerDtos,
    ExportTarget exportTarget = ExportTarget.Csv)
{
    return _buildersProvider.GetExportFileBuilder(exportTarget).GenerateOutput(selectedOptions, formCollectionAnswerDtos);
}

private void AppendSelectItem(ICollection<FormCollectionSelectItemDto> selectItems, FormBaseDto formBaseDto)
{
    var existingSelectItemDto = GetSelectItemWithExistingForm(selectItems, formBaseDto);
    if (existingSelectItemDto != null)
    {
        if (IsFormDynamic(existingSelectItemDto.Form))
        {
            AppendSelectItemControlsToExistingDynamicForm(existingSelectItemDto, formBaseDto);
        }
    }
    else
    {
        var selectItem = CreateSelectItem(formBaseDto);
        selectItems.Add(selectItem);
    }
}

private FormCollectionSelectItemDto CreateSelectItem(FormBaseDto formBaseDto)
{
    var selectItem = new FormCollectionSelectItemDto
    {
        FormId = formBaseDto.Id,
```

```
            Form = formBaseDto,
            ControlSelectItems = CreateSelectItemControls(formBaseDto)
        };

        return selectItem;
    }

    private List<ControlSelectItem> CreateSelectItemControls(FormBaseDto formBaseDto)
    {
        var controlSelectItems = new List<ControlSelectItem>();
        if (IsFormDynamic(formBaseDto))
        {
            foreach (var currentControl in ((DynamicFormDto) formBaseDto).Controls)
            {
                if (_ignorableControlTypes.Contains(currentControl.Type))
                {
                    continue;
                }

                var newControlSelectItem = new ControlSelectItem
                {
                    Label = currentControl.Label,
                    ControlId = currentControl.Id
                };

                controlSelectItems.Add(newControlSelectItem);
            }
        }
        else if (IsFormStatic(formBaseDto))
        {
            foreach (var currentControl in ((StaticFormDto) formBaseDto).Values)
            {
                var newControlSelectItem = new ControlSelectItem
                {
                    Label = currentControl.Key,
                    IsStatic = true
                };

                controlSelectItems.Add(newControlSelectItem);
            }
        }

        return controlSelectItems;
    }

    private void AppendSelectItemControlsToExistingDynamicForm(FormCollectionSelectItemDto existingSelectItemDto,
```

```
        FormBaseDto formBaseDto)
    {
        var existingDynamicFormDto = (DynamicFormDto) existingSelectItemDto.Form;
        var currentDynamicFormDto = (DynamicFormDto) formBaseDto;

        foreach (var currentControl in currentDynamicFormDto.Controls)
        {
            if (ControlExistsInDynamicForm(existingDynamicFormDto, currentControl) ||
_ignorableControlTypes.Contains(currentControl.Type))
            {
                continue;
            }

            var newControlSelectItem = new ControlSelectItem
            {
                Label = currentControl.Label,
                ControlId = currentControl.Id
            };

            existingSelectItemDto.ControlSelectItems.Add(newControlSelectItem);
        }
    }

    private bool ControlExistsInDynamicForm(DynamicFormDto existingDynamicFormDto, Control currentControl)
    {
        return existingDynamicFormDto.Controls.Any(existingControl => existingControl.Id == currentControl.Id);
    }

    private bool IsFormDynamic(FormBaseDto formBaseDto)
    {
        return formBaseDto.Type == FormType.Dynamic;
    }

    private bool IsFormStatic(FormBaseDto formBaseDto)
    {
        return formBaseDto.Type == FormType.Static;
    }

    private FormCollectionSelectItemDto GetSelectItemWithExistingForm(
        ICollection<FormCollectionSelectItemDto> selectItems, FormBaseDto formBaseDto)
    {
        return selectItems.FirstOrDefault(selectItem => selectItem.FormId == formBaseDto.Id);
    }
  }
}
```

FormController

```
using System;
using System.Threading.Tasks;
using AutoMapper;
using FormsGenerator.Domain.Entities;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Models.Filtering;
using Microsoft.AspNetCore.Mvc;

namespace FormsGenerator.Web.Controllers
{
   [Route("api/forms")]
   public class FormController : Controller
   {
      private readonly IFormService _formService;
      private readonly IMapper _mapper;

      public FormController(
         IFormService formService,
         IMapper mapper)
      {
         _formService = formService;
         _mapper = mapper;
      }

      [HttpGet]
      public async Task<IActionResult> Get([FromQuery] FormFilter filters)
      {
         // add model for filter

         var forms = await _formService.GetAll(filters);
         return Ok(forms);
      }

      [HttpGet("{id}")]
      public async Task<IActionResult> Get(Guid id)
      {
         var form = await _formService.GetById(id);
         return Ok(form);
      }

      [HttpGet("name/{formName}")]
      public async Task<IActionResult> Get(string formName)
      {
```

```
        var form = await _formService.GetByName(formName);
        if (form == null)
        {
            return NotFound();
        }

        return Ok(form);
    }

    [HttpGet("{id}/inUse")]
    public async Task<IActionResult> InUse(Guid id)
    {
        var inUse = await _formService.IsFormInUse(id);
        if (!inUse)
        {
            return NotFound();
        }

        return Ok();
    }

    [HttpPost]
    public async Task<IActionResult> Create([FromBody] DynamicFormDto formDto)
    {
        var form = _mapper.Map<DynamicFormDto, DynamicForm>(formDto);
        var createdForm = await _formService.Create(form);
        return Ok(createdForm);
    }

    [HttpPut]
    public async Task<IActionResult> Update([FromBody] DynamicFormDto formDto)
    {
        var form = _mapper.Map<DynamicFormDto, DynamicForm>(formDto);
        var updatedForm = await _formService.Update(form);
        return Ok(updatedForm);
    }

    [HttpDelete]
    public async Task<IActionResult> Delete(Guid id)
    {
        await _formService.Delete(id);
        return Ok();
    }
  }
}
```

FormCollectionController

```
using System;
using System.Threading.Tasks;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Models.Filtering;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FormsGenerator.Web.Controllers
{
   [Route("api/formCollections")]
   public class FormCollectionController : Controller
   {
      private readonly IFormCollectionService _formCollectionService;

      public FormCollectionController(IFormCollectionService formCollectionService)
      {
         _formCollectionService = formCollectionService;
      }

      [AllowAnonymous]
      [HttpGet]
      public async Task<IActionResult> Get([FromQuery] FormCollectionFilter filters)
      {
         // add model for filter

         var formCollections = await _formCollectionService.GetAll(filters);
         return Ok(formCollections);
      }

      [AllowAnonymous]
      [HttpGet("{id}")]
      public async Task<IActionResult> Get(Guid id)
      {
         var formCollection = await _formCollectionService.GetById(id);
         if (formCollection == null)
         {
            return NotFound();
         }

         return Ok(formCollection);
      }

      [HttpGet("name/{formCollectionName}")]
```

```
    public async Task<IActionResult> GetByName(string formCollectionName)
    {
        var formCollection = await _formCollectionService.GetByName(formCollectionName);
        if (formCollection == null)
        {
            return NotFound();
        }

        return Ok(formCollection);
    }

    [HttpPost]
    public async Task<IActionResult> Create([FromBody] FormCollectionDto formCollection)
    {
        await _formCollectionService.Create(formCollection);

        return Ok();
    }

    [HttpPut]
    public async Task<IActionResult> Update([FromBody] FormCollectionDto formCollection)
    {
        await _formCollectionService.Update(formCollection);

        return Ok();
    }

    [HttpDelete]
    public async Task<IActionResult> Delete(Guid id)
    {
        await _formCollectionService.Delete(id);
        return Ok();
    }
  }
}
```

FormCollectionAnswerController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using FormsGenerator.Services.Contracts.DomainServices;
using FormsGenerator.Services.Contracts.Dtos;
using FormsGenerator.Services.Contracts.Exceptions;
```

```
using FormsGenerator.Web.Utility;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FormsGenerator.Web.Controllers
{
   [Route("api/formCollectionAnswers")]
   public class FormCollectionAnswerController : Controller
   {
      private readonly IFormCollectionAnswerService _formCollectionAnswerService;

      public FormCollectionAnswerController(IFormCollectionAnswerService formCollectionAnswerService)
      {
         _formCollectionAnswerService = formCollectionAnswerService;
      }

      [HttpGet]
      public async Task<IActionResult> Get()
      {
         var formCollectionAnswerDto = await _formCollectionAnswerService.GetAll();
         if (formCollectionAnswerDto == null)
         {
            return NotFound();
         }

         return Ok(formCollectionAnswerDto);
      }

      [AllowAnonymous]
      [HttpGet("{id}")]
      public async Task<IActionResult> Get(Guid id)
      {
         var formCollectionAnswerDto = await _formCollectionAnswerService.GetById(id);
         if (formCollectionAnswerDto == null)
         {
            return NotFound();
         }

         return Ok(formCollectionAnswerDto);
      }

      [HttpGet("{collectionId}/answers")]
      public async Task<IActionResult> GetByCollectionId(Guid collectionId)
      {
         var formCollectionAnswerDto = await _formCollectionAnswerService.GetAllByCollectionId(collectionId);
         if (formCollectionAnswerDto == null)
```

```
        {
            return NotFound();
        }

        return Ok(formCollectionAnswerDto);
    }

    [AllowAnonymous]
    [HttpPost("{collectionId}")]
    public async Task<IActionResult> Create(Guid collectionId)
    {
        var clientId = GetClientId(User.Claims);

        FormCollectionAnswerDto formCollectionAnswerDto;

        try
        {
          formCollectionAnswerDto =
            await _formCollectionAnswerService.Create(collectionId, clientId);

            formCollectionAnswerDto.UserAgent = Request.Headers["User-Agent"];
        }
        catch (ArgumentNullException exception)
        {
            return BadRequest(exception.Message);
        }
        catch (AnswerAlreadyExistsException exception)
        {
            return BadRequest(exception.Message);
        }

        return Ok(formCollectionAnswerDto);
    }

    [AllowAnonymous]
    [HttpPost("share")]
    public async Task<IActionResult> Share([FromBody] FormCollectionAnswerDto formCollectionAnswerDto)
    {
        var responseMessage = await _formCollectionAnswerService.Share(formCollectionAnswerDto);
        if (responseMessage.IsSuccessStatusCode)
        {
            return Ok();
        }

        return BadRequest(responseMessage.ReasonPhrase);
    }
```

```
    [AllowAnonymous]
    [HttpPut]
    public async Task<IActionResult> Update([FromBody] FormCollectionAnswerDto formCollectionAnswer)
    {
        formCollectionAnswer.UserAgent = Request.Headers["User-Agent"];

        var updatedFormCollectionAnswerDto = await _formCollectionAnswerService.Update(formCollectionAnswer);

        return Ok(updatedFormCollectionAnswerDto);
    }

    [HttpDelete]
    public async Task<IActionResult> Delete(Guid id)
    {
        await _formCollectionAnswerService.Delete(id);
        return Ok();
    }

    private Guid? GetClientId(IEnumerable<Claim> claims)
    {
        var clientIdClaim = claims.FirstOrDefault(claim => claim.Type == Constants.JwtUidClaimType);

        return clientIdClaim == null ? (Guid?) null : Guid.Parse(clientIdClaim.Value);
    }
  }
}
```

Додаток В
«Специфікація»

ГЮИК.509000.005

(позначення документу)

Код	Назва	Примітки
	Документація	
ГЮИК.509000.005 ПЗ	Пояснювальна записка	77 стор.
ГЮИК.509000.005 -01 12 01	Текст програми	21 стор.

Зм.	Лист	№ докум	Підп.	Дата	ГЮИК.509000.005			
Розроб.		ДемченкоО. Е..			*Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних*		Лист	Листів
Перевірив.		Ситніков Д. Е..					*1*	*1*
Н. Контр.		Ситніков Д. Е..			Специфікація	*ХНУРЕ Кафедра СТ*		
Затверд.		Гребеннік І.В.						

Додаток Г
«Відомість атестаційної роботи»

ГЮИК.509000.005 Д3
(позначення документу)

№	Позначення	Назва	Дод. відомості
		Текстові документи	
1.	ГЮИК.509000.005 ПЗ	Пояснювальна записка	77 стор.
2.	ГЮИК.509000.005 -01 12 01	Текст програми	21 стор.
3.	ГЮИК.509000.005	Специфікація	1 стор.
		Графічні документи	
4.	ГЮИК.509000.005	Алгоритм Apriori	1 аркуш
5.		Алгоритм створення форми	1 аркуш
6.		Алгоритм створення колекції форм	1 аркуш
7.		Алгоритм відповіді на колекцію форм	1 аркуш
8.		Алгоритм запуску аналізу колекції форм	1 аркуш
9.		Діаграма прецедентів для користувача «Гість»	1 аркуш
10.		Діаграма прецедентів для користувача	1 аркуш
		«Користувач»	
11.		Діаграма прецедентів для користувача	1 аркуш
.		«Адміністратор»	
12.		Узагальнена схема роботи Web API	1 аркуш
13.		Час прогону алгоритма Apriori	1 аркуш
14.		Використання CPU алгоритмом Apriori	1 аркуш
15.		Використання оперативної памяті алгоритмом	1 аркуш
		Apriori	
17.			
18.			
.			

Зм.	Лист	№ докум	Підп.	Дата	ГЮИК.509000.005 ДЗ		
Розроб.		Демченко О. Е.			Розробка методу використання технологій Big Data та Data Mining в інтелектуальних системах обробки неструктурованих даних	Лист	Листів
Перевірив.		Ситніков Д. Е..				1	1
Н. Контр.		Ситніков Д. Е..				ХНУРЕ Кафедра СТ	
Затверд.		Гребеннік І.В.					